SORCERY BREWS
This manual is a treasury of programming tricks that are specific to the Exidy Sorcerer computer, although much is applicable to other microcomputers that employ either Microsoft Basic or a Z80 microprocessor. With this ready reference of valuable examples at your fingertips, your programming efforts will be greatly simplified, and your programs will be more professional in both appearance and performance. Using this manual will unleash the hidden powers of your Sorcerer. You will graduate from being an apprentice to being a full wizard as you study and use the brews concocted by masters of the Sorcerer.
# TABLE OF CONTENTS

## CHAPTER 1  ---  BASIC PROGRAM TIPS

1.01 RELOCATE BASIC PROGRAMS TO A DIFFERENT RUN ADDRESS  
1.02 END-OF-PROGRAM ADDRESS FOR ROMPAC BASIC  
1.03 CSAVE BASIC AND MACHINE LANGUAGE ROUTINES TOGETHER  
1.04 SAVE A CHARACTER SET AND A BASIC PROGRAM TOGETHER  
1.05 AUTO-EXECUTE BASIC PROGRAMS  
1.06 FIND TOP-OF-MEMORY IN ADDRESSES F000-F001 HEX  
1.07 ARRAY SPACE REQUIREMENT  
1.08 CONCEAL 'REM' AND LINE NUMBER IN LISTINGS  
1.09 MACHINE CODE ROUTINES HIDDEN IN BASIC REMARKS  
1.10 THREE MEMORY SIZE DEPENDENT BYTES IN THE BASIC WORK AREA  
1.11 AN IDEAL PROGRAM START  
1.12 FASTER BASIC EXECUTION  
1.13 MERGE TWO BASIC PROGRAMS TOGETHER

## CHAPTER 2  ---  BASIC COMMANDS

2.01 CLEAR COMMAND  
2.02 RESTORE nnnn WILL RESTORE TO A SPECIFIC LINE NUMBER  
2.03 SWAP INTEGER VARIABLES  
2.04 LEN(STR$(1)) IS 2 BECAUSE A SPACE PRECEDES THE DIGIT  
2.05 MID$ FUNCTION EXAMPLES  
2.06 USE CHR$(64) TO PRINT THE '@' SIGN  
2.07 ORDER OF PRECEDENCE FOR NUMERICAL OPERATIONS  
2.08 FLOATING POINT INACCURACIES  
2.09 NUMERICAL RELATIONS  
2.10 LOGICAL OPERATOR TRUTH TABLES  
2.11 ONE SECOND DELAY LOOP IN BASIC  
2.12 A 0.81 SECOND DELAY LOOP IN THE MONITOR  
2.13 A 3.24 SECOND DELAY LOOP IN THE MONITOR  
2.14 MULTIPLE LINE CHARACTER STRINGS  
2.15 PRINT DECIMAL NUMBER IN HEXADECIMAL  
2.16 PRINT DECIMAL NUMBER IN HEXADECIMAL  
2.17 PRINT DECIMAL NUMBER IN HEXADECIMAL  
2.18 CONVERT HEXADECIMAL TO DECIMAL  
2.19 CONVERT DECIMAL TO HEXADECIMAL  
2.20 MENU SELECTION UTILIZING 'IF X THEN RUN' STATEMENT  
2.21 CARTESIAN COORDINATES FROM POLAR EQUATIONS  
2.22 SHELL SORT SUBROUTINE TO PUT ARRAY Z() IN ASCENDING ORDER
CHAPTER 3 --- FUNCTIONS
3.01 RANDOM NUMBER FUNCTION
3.02 RANDOMIZE FUNCTION
3.03 INT FUNCTION EXAMPLES
3.04 SGN() FUNCTION
3.05 DEF FUNCTION RULES
3.06 DECIMAL POINT ALIGNMENT
3.07 TABULATED PRINTING OF DOLLARS AND CENTS
3.08 ROUND OFF FUNCTION
3.09 ASCII CODE FOR A HEX DIGIT
3.10 FIX COMMAND
3.11 BASE 10 LOGORITHM
3.12 PI = 3.14159 = 355/113

CHAPTER 4 --- KEYBOARD
4.01 ACCEPT 'YES' ANSWER WITH INPUT OF 'Y', 'YE' OR 'YES'
4.02 KEYBOARD SCAN OR 'GET' COMMAND
4.03 PRINT ON LINE AFTER INPUT STATEMENT
4.04 INPUT STRINGS CONTAINING COMMAS AND '@' SYMBOLS
4.05 KEYBOARD SCAN FOR 'GRAPHIC', 'CONTROL', AND 'SHIFT'
4.06 WAIT FOR USER TO PRESS 'SHIFT' KEY TO CONTINUE

CHAPTER 5 --- VIDEO
5.01 DOUBLE SPACE LISTINGS
5.02 SCREEN POKE ADDRESS FOR ANY ROW AND COLUMN
5.03 MAKE THE CURSOR DISAPPEAR AFTER A PRINT STATEMENT
5.04 REMOVE CURSOR FROM SCREEN
5.05 SCREEN ADDRESSING
5.06 SYNC SCREEN MOTION WITH VIDEO HORIZONTAL SYNC PULSE
5.07 PRINT AT SUBROUTINE
5.08 PLACE CURSOR AT ROW AND COLUMN
5.09 PLACE CURSOR AT ROW AND COLUMN
5.10 DRAW BOX ROUTINE
5.11 CREATE INVERSE VIDEO CHARACTER SET
5.12 CREATE DOUBLE WIDE CHARACTER SET

CHAPTER 6 --- JOYSTICKS
6.01 JOYSTICK / KEYBOARD STANDARD FOR THE SORCERER
6.02 JOYSTICK INTERFACE STANDARD TO PARALLEL PORT
6.03 JOYSTICK CIRCUIT DIAGRAM
6.04 JOYSTICK EXAMPLE USING BASIC STATEMENTS
6.05 JOYSTICK EXAMPLE USING MACHINE LANGUAGE SOFTWARE
6.06 MACHINE LANGUAGE JOYSTICK ROUTINE
CHAPTER 7 --- SOUND

7.01 PRINCIPLE OF ONE VOICE SOUND
7.02 ONE VOICE MUSIC ROUTINE
7.03 PIEZO SPEAKER AUDIO PROMPTER
7.04 CONTROL 'G' BEEP

CHAPTER 8 --- BASIC ROMPAC

8.01 BASIC ROMPAC MAP
8.02 BASIC ROMPAC WORK AREA

CHAPTER 9 --- MONITOR

9.01 MONITOR POKE ADDRESSES
9.02 MONITOR WORKAREA
9.03 USEFUL MONITOR ROUTINES AND THEIR FUNCTIONS
9.04 RELOCATE THE MONITOR STACK
9.05 ERASE MEMORY BY FILLING IT WITH ZEROES
9.06 EXECUTE MONITOR COMMANDS FROM A BASIC PROGRAM

CHAPTER 10 --- MACHINE LANGUAGE INTERFACE

10.01 PROTECT MEMORY FOR MACHINE LANGUAGE ROUTINES
10.02 CALL A MACHINE LANGUAGE ROUTINE
10.03 FASTER USR() PARAMETER PASSING
10.04 POKE MACHINE HEX CODE INTO MEMORY
10.05 PASS ARGUMENT IN 'A' REGISTER TO MACHINE LANGUAGE ROUTINE
10.06 PASS TWO PARAMETERS USING THE OUT I,J INSTRUCTION
10.07 PASS ROUTINE ADDRESS IN USR() FUNCTION CALL
10.08 MULTIPLE USR() ROUTINES SELECTED BY THE LETTER IN ()
10.09 UP-LOADER FOR MACHINE LANGUAGE ROUTINES

CHAPTER 11 --- MACHINE CODE ROUTINES

11.01 ROW - COLUMN ROTATION OF A CHARACTER CELL
11.02 IMAGE DRIVER TO CONVERT TO >SA FILES.
11.03 PINE WOOD DERBY CONTROLLER
11.04 PINE WOOD DERBY'S BASIC PROGRAM
11.05 IDEAL MACHINE LANGUAGE PROGRAM START
CHAPTER 12 --- I/O DRIVERS

12.01 RS232 OUTPUT DRIVER ROUTINE
12.02 RS232 DRIVER WITH PERFORATION SKIP CONTROL
12.03 RS232 INPUT DRIVER ROUTINE
12.04 DUMB TERMINAL ROUTINE
12.05 VARIABLE LINE LENGTHS FOR PRINTERS
12.06 CENTRONICS SCREEN PRINT ROUTINE
12.07 CENTRONICS PRINTER DRIVER
12.08 ACCESS CENTRONICS PRINTER DRIVER FROM BASIC
12.09 PROGRAMMING THE UART FOR PARITY OPTIONS

CHAPTER 13 --- CASSETTE TAPE

13.01 WRITE DATA TO CASSETTE TAPE
13.02 READ DATA FROM CASSETTE TAPE

CHAPTER 14 --- EDITOR FOR BASIC

14.01 EDITOR FOR BASIC INSTRUCTIONS
14.02 EDITOR FOR BASIC SOURCE LISTING

CHAPTER 15 --- CP/M

15.01 SAVE ROMPAC BASIC PROGRAM ON CP/M DISK
15.02 SAVE WORD PROCESSOR ROMPAC FILES ON CP/M DISK
15.03 CP/M COMMANDS
15.04 BIOS MODIFICATION TO GIVE BACKSPACE TYPE RUBOUTS
15.05 CP/M LOAD ROUTINE OF WP FILES FOR DEVPAC

CHAPTER 16 --- WORD PROCESSOR

16.01 WORD PROCESSOR PRINTER DRIVER
16.02 SALVAGE WORD PROCESSOR FILE FROM RESET
16.03 PRINTER DRIVER TO SEND ESCAPE SEQUENCES

CHAPTER 17 --- DEVELOPMENT PAC

17.01 PAUSE DEVELOPMENT PAC LISTINGS
17.02 MEMORY PARTITIONS FOR 32K CONFIGURATION
17.03 I/O VECTOR ASSIGNMENTS
17.04 SAMPLE COMMANDS

CHAPTER 18 --- PLOTTING

18.01 BEAUTIFUL BIRTHDAY PLOTS
18.02 3-D FUNCTION PLOTS USING SHADED PRINT DENSITY
### CHAPTER 19 --- TABLES AND FORMS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.01</td>
<td>BASIC'S TOKENS</td>
</tr>
<tr>
<td>19.02</td>
<td>HEX - BINARY CONVERSION TABLE</td>
</tr>
<tr>
<td>19.03</td>
<td>POWERS OF 2 TABLE</td>
</tr>
<tr>
<td>19.04</td>
<td>DECIMAL - HEXADECIMAL CONVERSION TABLE</td>
</tr>
<tr>
<td>19.05</td>
<td>CONTROL CODES AND THEIR FUNCTIONS</td>
</tr>
<tr>
<td>19.06</td>
<td>GRAPHIC CHARACTER DESIGNER'S FORM</td>
</tr>
<tr>
<td>19.07</td>
<td>MACHINE LANGUAGE CODING FORM</td>
</tr>
<tr>
<td>19.08</td>
<td>SERIAL INTERFACE PINOUTS</td>
</tr>
<tr>
<td>19.09</td>
<td>PARALLEL INTERFACE PINOUTS</td>
</tr>
<tr>
<td>19.10</td>
<td>DATA PORT ASSIGNMENTS</td>
</tr>
<tr>
<td>19.11</td>
<td>ASCII CHARACTER SET</td>
</tr>
<tr>
<td>19.12</td>
<td>KEYBOARD STRUCTURE</td>
</tr>
<tr>
<td>19.13</td>
<td>GRAPHIC CHARACTER STARTING ADDRESSES</td>
</tr>
<tr>
<td>19.14</td>
<td>ONE STROKE BASIC COMMANDS</td>
</tr>
</tbody>
</table>
FORWARD

"In promulgating your esoteric cognitions, or articulating your superficial sentimentalities and amicable, philosophical or psychological observations, beware of platitudinous ponderosity. Let your conversational communications possess a clarified conciseness, a compact comprehensibleness, coalescent consistency, and a concantenated cogency. Eschew all conglomerations of flatulent garrulity, jejune babblement and asinine affectation. Let your extemporaneous and unpremeditated expatiations have intelligibility and vivacious vivacity, without rodomontade and thrasonical bombast. Sedulously avoid all polysyllabic profundity, pompous prolixity, osittaceous vacuity, ventriloquial verbosity, and veniloquent vapidity. Shun double-entendres, pruvient jocosity, and pestiferous profanity, obscurant or apparent. In other words, write plainly, briefly, naturally, sensibly, truthfully, purely. Keep from complexities; don't put on airs; write what you mean; and don't forget that others are trying to understand you. Simply stated -- Reread this paragraph until you understand the veniloquent vapidity."

I've tried to heed this good counsel throughout this manual.

To my patient wife

MARILYN
CHAPTER 1 --- PROGRAM TIPS

1.01 RELOCATE PROGRAMS TO A DIFFERENT RUN ADDRESS.

BYE
>EN 149
0149: 00 10 /
>EN OFFF
OFFF: 0 /
>PP

PROGRAM CAN BE RUN AND CSAVED IN NORMAL FASHION.
TO CLOAD, THOUGH, 0149 HEX MUST BE ALTERED AS ABOVE.
THIS ALLOWS A BASIC PROGRAM TO HAVE A PROTECTED MEMORY
SPACE FOR MACHINE LANGUAGE ROUTINES FROM 01D5H TO ONE
BYTE BEFORE THE START OF YOUR RELOCATED BASIC PROGRAM.

1.02 END-OF-PROGRAM ADDRESS FOR ROMPAC BASIC.

BYE
>DU 1B7 1B8

ADDRESS STORED IN THESE TWO BYTES.

ADDR 0 1 2 3 4 5 6 7 8
01B0: .. .. yy xx

HEX ADDRESS IS STORED IN REVERSE BYTE ORDER IN ADDRESS
01B7 AND 01B8 HEX. ADDRESS IS READ AS xxyy.

1.03 CSAVE BASIC AND MACHINE LANGUAGE ROUTINES.

PLACE MACHINE LANGUAGE ROUTINES IN MEMORY JUST BEYOND
THE END OF A BASIC PROGRAM.

PUT THE ADDRESS TO SAVE THROUGH IN BYTES 1B7 AND 1B8.

EXAMPLE: IF MACHINE CODE OCCUPIES TO ADDRESS 264F HEX.

BYE
>EN 1B7
01B7: 4F 26 /
>PP

WHEN THE BASIC PROGRAM AND THE MACHINE CODE RELOAD,
THE MACHINE CODE NEEDS TO BE MOVED BACK TO ITS ORIGINAL
ADDRESS IN MEMORY. BE CAREFUL THAT VARIABLE STORAGE
DOES NOT OVERWRITE IT WHEN THE BASIC PROGRAM RUNS.
1.04 SAVE A CHARACTER SET AND A BASIC PROGRAM TOGETHER.

BYE
>DU 1B7 1B8              FIND END OF PROGRAM ADDRESS.
>SA XMPLE FC00 xxxyy     SAVE THROUGH xxxyy FROM 1B7 1B8.
>LO XMPLE                TO RERUN YOUR PROGRAM.
>PP
RUN

=====================================================================

1.05 AUTO-EXECUTE BASIC PROGRAMS

PROGRAM MUST HAVE A LINE 0.  EXAMPLE:  0 REM
FIND PROGRAM END IN 1B7-1B8 HEX FOR xxxyy ADDRESS.

>SE X=C858     (AUTO EXECUTE ADDRESS)
>SE F=4D       (FILE TYPE)
>SA PROG 1B7 xxxyy

LOAD AND AUTO EXECUTE WITH    >LOG

=====================================================================

1.06 FIND TOP-OF-MEMORY IN ADDRESSES F000-F001 HEX.

100 RAMSIZE = PEEK(-4095) * 256 + PEEK(-4096)

=====================================================================

1.07 ARRAY SPACE REQUIREMENT

THE NUMBER OF BYTES REQUIRED TO STORE AN ARRAY IS:

( # OF ELEMENTS ) * 4 + ( # OF DIMENSIONS ) * 2 + 6

EXAMPLE:  100 DIM A(10,10) : REM USES 494 BYTES
           : REM 494 = 11*11*4+2*2+6

=====================================================================

1.08 CONCEAL 'REM' AND LINE NUMBER IN SCREEN LISTINGS.

ONE CAN CONCEAL THE LINE NUMBER OF REMARK STATEMENTS BY USING THE FOLLOWING TRICK.  THE APPEARANCE OF REMARK STATEMENTS IMPROVES BECAUSE THEY READILY STAND OUT.
SAMPLE OUTPUT:

100 PRINT "THIS IS A SAMPLE SCREEN LISTING"

--- A REMARK, BUT NO 'REM' OR LINE # SHOWS ---

120 PRINT "LINE NUMBER 110 IS PRESENT, YET HIDDEN"

TO DO THE ABOVE TO A REMARK STATEMENT, DO THE FOLLOWING:

1. TYPE THE LINE NUMBER AND 'REM', IE. 110 REM

2. NOW, PRESS THE CURSOR LEFT KEY SEVEN OR MORE TIMES TO MOVE THE CURSOR ALL THE WAY BACK TO THE LEFT MARGIN.

3. USE THE SPACE BAR TO ERASE THE VISIBLE LINE NUMBER AND THE 'REM' LETTERS.

4. NOW, PRESS 'LINE FEED' TO INSERT A BLANK LINE.

5. TYPE THE REMARK STATEMENT'S TEXT.

6. TYPE ANOTHER 'LINE FEED' TO INSERT A BLANK LINE BELOW.

7. FINALLY, TERMINATE THE LINE WITH CARRIAGE RETURN.

================================================================

1.09 MACHINE CODE ROUTINES HIDDEN IN BASIC REMARKS

ONE CAN STORE A SHORT MACHINE LANGUAGE ROUTINE AS PART OF BASIC STATEMENT BY PUTTING IT IN A REM STATEMENT. IT WILL AUTOMATICALLY BE SAVED AND LOADED WITH THE BASIC PROGRAM.

EXAMPLE:

100 REM < A LINE FULL OF SPACES >
110 REM THE REST OF THE PROGRAM FOLLOWS

IN MEMORY THE FIRST LINE LOOKS LIKE:

01D5: yy xx 64 00 8F
01DA: 20 20 20 20 20 20 20 20 20 20 20 20 etc.

THE xxxyy IS THE LINK POINTER ADDRESS OF WHERE THE NEXT LINE STARTS. THE LINE NUMBER OF 100 IS STORED AS 64 00. THE 'REM' COMMAND IS STORED AS 8F. STARTING IN ADDRESS 01DA ARE ALL THE SPACES OF THE REM TEXT.

YOU CAN INSERT A SHORT MACHINE LANGUAGE ROUTINE IN THE ADDRESS SPACE BETWEEN 01DAH AND xxxy, FOR EXAMPLE. ONCE THE CODE IS INSERTED, DON'T ALTER THE REM STATEMENT OR ANY LINE WHICH MIGHT PRECEDE THE REM STATEMENT.

ALSO, THE CODE INSERTED CANNOT CONTAIN A 00 BYTE AS BASIC WILL TREAT IT AS AN END-OF-LINE MARKER.
1.10 THREE MEMORY SIZE DEPENDENT BYTES IN THE BWA

0145-6H - POINTER TO TOP OF BASIC STACK.
0192-3H - POINTER TO HIGHEST RAM LOCATION.
01A6-7H - POINTER TO TOP OF FREE STRING SPACE.

IF THE BASIC WORK AREA FROM 0100H THROUGH 01D4H IS SAVED WITH A BASIC PROGRAM IN THIS FASHION: >SA PROG 100 xxxyy, THE PROGRAM WILL NOT RUN ON A MACHINE WHICH HAS LESS MEMORY THAN THE MACHINE ON WHICH THE PROGRAM WAS RECORDED.

THE CONFLICT CAN BE OVERCOME BE LOADING BYTES 0146H, 0193H, AND 01A7H WITH THE CORRECT VALUE, IE. THE DEFAULT VALUE LOADED BY BASIC ON POWER UP. THE VALUE IS 03EH FOR A 16K MACHINE, 07EH FOR A 32K MACHINE, AND 0BEH FOR A 48K MACHINE.

=================================================================================================

1.11 AN IDEAL PROGRAM START

0 REM --- PROGRAM NAME ---
100 :
110 REM AUTHOR'S COPYRIGHT NOTICE
120 REM PROGRAM DATE AND VERSION NUMBER
130 :
140 REM DEFINE CURSOR CONTROL VARIABLES
150 CL$=CHR$(1) : CP$=CHR$(12) : CH$=CHR$(17)
160 CR$=CHR$(19) : CU$=CHR$(23) : CD$=CHR$(26)
170 :
180 REM COMPUTE MEMORY SIZE TO LOCATE MONITOR WORK AREA
190 MS=256*PEEK(-4095)+PEEK(-4096)
200 IF MS>32767 THEN MS=MS-65536
210 :
220 REM LOCATE MONITOR WORK AREA CONTROL BYTES
230 PC=MS-47 :REM PC is printer control poke address.
240 SS=MS-48 :REM SS is screen speed poke address.
250 BR=MS-49 :REM BR is baud rate poke address.

===============================================================================================

1.12 FASTER BASIC EXECUTION

BASIC STARTS AT THE BEGINNING OF A PROGRAM AND SCANS THROUGH LOOKING FOR THE LINE NUMBER REFERENCED BY A GOSUB, IF-THEN, OR GOTO.

OPTIMIZE BY HAVING OFTEN CALLED LINE NUMBERS NEAR THE BEGINNING OF A PROGRAM. HAVE Seldom used lines, such as PROGRAM INSTRUCTIONS AND INITIALIZATION, AT THE BOTTOM OF A PROGRAM.
1.13 MERGE TWO BASIC PROGRAMS TOGETHER.

1. THE LINE NUMBERS IN THE PROGRAM TO BE MERGED MUST BE GREATER THAN THE LAST LINE NUMBER IN THE FIRST PROGRAM. USE A LINE RENUMBERING PROGRAM IF NECESSARY TO ACCOMPLISH THIS.

2. LOAD THE FIRST PROGRAM INTO MEMORY. FIND ITS END LOCATION IN ADDRESS 01B7 AND 01B8 HEX.

3. USE THE END ADDRESS TO FIND THE THREE ZEROS IN MEMORY MARKING THE END OF THE PROGRAM. WRITE DOWN THE ADDRESS OF THE SECOND ZERO.

4. USE THE MONITOR TO LOAD THE PROGRAM TO BE MERGED. USE >LO FILENAME 1 xxyy WHERE xxyy IS THE ADDRESS OF THE SECOND ZERO FROM STEP #3.

5. ADD THE MERGING PROGRAM'S SIZE TO xxyy TO HELP LOCATE THE THREE ZEROS AT THE END OF THE TWO COMBINED PROGRAMS.

6. STORE THE ADDRESS OF THE THIRD ZERO IN BYTES 01B7 AND 01B8 HEX. THE ADDRESS IS STORED IN REVERSE BYTE ORDER.

7. IN BASIC, ADD AND DELETE A LINE NUMBER 0, IE. 0 REM THIS WILL RECOMPUTE ALL OF THE LINK POINTERS. THE TWO PROGRAMS ARE NOW MERGED AND CAN BE SAVED ON TAPE.
CHAPTER 2 --- BASIC COMMANDS

2.01 CLEAR COMMAND

100 CLEAR xx, yy

xx IS THE NUMBER OF BYTES TO RESERVE FOR STRING SPACE. 
yy IS THE TOP-OF-MEMORY ADDRESS FOR THIS STRING SPACE.

EXAMPLE: 100 CLEAR 500, 24576

500 BYTES ARE RESERVED FOR STRING MANIPULATIONS. 
The STRING WORKSPACE WILL START AT 6000 HEX AND GROW 
DOWNWARD IN MEMORY TOWARD THE VARIABLE STORAGE SPACE.

MEMORY FROM 6000 HEX TO THE BOTTOM OF THE MONITOR STACK 
WORKSPACE IS NOW 'PROTECTED' MEMORY AVAILABLE FOR 
MACHINE LANGUAGE ROUTINES.

===============================================================

2.02 RESTORE nnnn WILL RESTORE TO A SPECIFIC LINE #.

===============================================================

2.03 SWAP INTEGER VARIABLES

100 A = A XOR B
B = B XOR A
A = A XOR B

===============================================================

2.04 LEN(STR$(1)) = 2 SINCE A SPACE PRECEDES THE DIGIT. 
USE A$ = MID$(STR$(J),2) TO EXTRACT JUST THE DIGITS.

==================================================================

2.05 MID$ FUNCTION EXAMPLES.

100 A$ = "1234567890"
110 PRINT MID$(A$, 3, 2) : REM OUTPUT "34"
120 PRINT MID$(A$, 3, 4) : REM OUTPUT "3456"
130 PRINT MID$(A$, 3) : REM OUTPUT "34567890"
140 PRINT MID$(A$, 8, 5) : REM OUTPUT "890"

MID$(A$, 1, J) = LEFT$(A$, J)
2.06 USE CHR$(64) TO PRINT THE '@' SIGN.

EXAMPLE: 100 PRINT "5 APPLES ";CHR$(64);" 15 CENTS EACH"

===============================================================

2.07 ORDER OF PRECEDENCE FOR NUMERICAL OPERATIONS

1. () EXPRESSIONS IN PARENTHESES.
2. ^ EXPONENTIATION.
3. - NEGATION
4. * / MULTIPLICATION AND DIVISION
5. + - ADDITION AND SUBTRACTION
6. NUMERICAL RELATIONS
   = EQUAL
   <> NOT EQUAL
   < LESS THAN
   > GREATER THAN
   <= LESS THAN OR EQUAL
   >= GREATER THAN OR EQUAL
7. NOT
8. AND
9. OR

===================================================================

2.08 FLOATING POINT INACCURACIES

100 J= 30*30 : X= 30^2 : PRINT J,X

BOTH SHOULD PRINT 900, HOWEVER THE SECOND NUMBER IS
PRINTED AS 900.001 BECAUSE ALL OPERATIONS EMPLOY ONLY SIX
DIGITS OF ACCURACY.  THUS, ONE MAY HAVE TO TEST FOR A
BOUNDED CONDITION RATHER THAN FOR EQUALITY.  FOR EXAMPLE:

110 IF X>899.997 AND X<900.003 THEN PRINT " X = 900 "

=====================================================================

2.09 NUMERICAL RELATIONS

IF A RELATION IS 'TRUE', A VALUE OF -1 IS RETURNED.
IF A RELATION IS 'FALSE', A VALUE OF 0 IS RETURNED.

EXAMPLE: 100 PRINT 1=2, 1<2, 2=2, "A"="A", "AB"="AC"

OUTPUT: 0 -1 -1 -1 0
2.10 LOGICAL OPERATOR TRUTH TABLES

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>J</th>
<th>I AND J</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>J</th>
<th>I OR J</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

EXAMPLE: J AND 255 = JUST THE LOWER BYTE
          THIS IS A QUICK WAY TO GET THE
          LOWER BYTE OF AN ADDRESS TO POKE.

15 AND 13 = 13   IN BINARY  1111 AND 1101 = 1101
-1 AND J = J    BECAUSE -1 IS ALL ONES.
5 OR 2 = 7      IN BINARY  0101 OR 0010 = 0111
5 AND 2 = 0
NOT 0 = -1      FALSE = 0   TRUE = -1
NOT J = -(J+1)  TWO'S COMPLEMENT.

2.11 ONE SECOND DELAY LOOP IN BASIC

100 FOR J=1 TO 500:NEXT J

2.12 A 0.81 SECOND DELAY LOOP IN THE MONITOR

100 POKE 260,39 : POKE 261,224
110 X=USR(0)  : REM 0.81 SEC DELAY WITH EACH CALL

THIS 'CASSETTE OFF' ROUTINE GIVES 0.81 SECONDS OF DELAY.

2.13 A 3.24 SECOND DELAY LOOP IN THE MONITOR

100 POKE 260,160 : POKE 261,226
110 X=USR(0)  : REM 3.24 SEC DELAY WITH EACH CALL
2.14 MULTIPLE LINE CHARACTER STRINGS.

Graphical figures which occupy three or four print lines and are a few characters wide may be created in a single string and printed with one print statement. A single print statement used to place the figure on the screen is much faster than printing the figure with multiple print statements.

Example:

```
100 A$ = "*****
       * *
       * *
       *****"
110 PRINT A$
```

The above program will print a box made from asterisks. There are hidden characters in line 100 that make the string A$ print the box on four lines.

A string begins and ends with quote marks, and may contain any characters. Our string contains the asterisks that are seen, plus 'line feed', 'cursor left' and 'space' characters.

The string consists of the following sequence of characters in the order pressed from the keyboard:

```
A$ = "*****<LF><CL><CL><CL><CL><CL>
     * *<LF><CL><CL><CL><CL><CL>
     * *<LF><CL><CL><CL><CL><CL>
     *****"
```

The string A$ is 38 characters in length.

<LF> is a line feed character.
<CL> is a cursor left character.

===================================================================

2.15 PRINT DECIMAL NUMBER IN HEXADECIMAL.

```
0000: ED 5B 07 01 ENTRY LD DE,(0107H) ;GET I # IN E
0004: 57                    LD D,A     ;GET J # IN D
0005: CD E8 E1              CALL OE1E8H ;PRINT HEX #
0008: C1                    POP BC      ;RELIEVE STACK
0009: C9                    RET
```

Routine must reside at address 0 for RST OH command.

```
100 POKE 262,199 : REM 'OUT I,J' BECOMES RST OH COMMAND.
110 INPUT X : REM X IS DECIMAL # TO PRINT IN HEX.
120 I=X : IF X>32767 THEN I=X-65536
130 OUT I AND 255, X/256 : REM HEX # IS PRINTED.
```
2.16 PRINT DECIMAL NUMBER IN HEXADECIMAL

0000: CD DO C7  ENTRY CALL 0C7D0H ;GET USR() #
0003: C3 E8 E1  JP  0E1E8H  ;PRINT HEX #

100 POKE 260,0 : POKE 261,0 : REM USR() ENTRY ADDRESS
110 X = USR(J)  : REM PRINT HEX VALUE OF J

2.17 PRINT DECIMAL NUMBER IN HEXADECIMAL.

0000 11 yy xx  ENTRY LD DE,nn  nn IN DECIMAL
0003 C3 E8 E1  JP  0E1E8H  PRINT HEX #

TO USE THE ABOVE MACHINE LANGUAGE ROUTINE, FIRST PLACE DECIMAL NUMBER TO CONVERT IN BYTES 1 AND 2.

EXAMPLE: 100 POKE 260,0 : POKE 261,0 : REM USR() ADDR
110 POKE 1,J AND 255 : POKE 2,J/256
120 X=USR(0) : REM PRINT HEX EQUIVALENT OF J

2.18 CONVERT HEXADECIMAL TO DECIMAL

100 DEF FNA(X) = (X AND 15) - 9 * (X > 64)
110 :
120 A$ = "FC07" : REM TYPICAL HEXADECIMAL # TO CONVERT
130 :
140 N=0 : FOR J=1 TO LEN(A$)
150 N=N*16 + FNA(ASC(MID$(A$,J))): NEXT J
160 :
170 PRINT A$;" HEXADECIMAL EQUALS ";N;" DECIMAL"

2.19 CONVERT DECIMAL TO HEXADECIMAL

100 N = 40000 : REM TYPICAL NUMBER TO CONVERT TO HEX
110 :
120 J = N : A$="" :REM A$ TO RECEIVE HEXADECIMAL DIGITS
130 I = J AND 15 : J = INT(J/16)
140 A$= CHR$(I+48 - 7*(I > 9)) + A$ : IF J>0 THEN 130
150 :
160 PRINT N;" DECIMAL EQUALS ";A$;" HEXADECIMAL"
2.20 MENU SELECTION USING 'IF X THEN RUN' STATEMENT

100 PRINT "MENU SELECTION"
110 PRINT " 1 - FIRST PROGRAM"
120 PRINT " 2 - SECOND PROGRAM"
130 PRINT " 3 - THIRD PROGRAM"
140 :
150 INPUT "ENTER SELECTION NUMBER ", X
160 IF X=1 THEN RUN 1000
170 IF X=2 THEN RUN 2000
180 IF X=3 THEN RUN 3000
190 PRINT "ENTER NUMBER FROM 1 TO 3, PLEASE" : GOTO 150
200 :
1000 REM --- FIRST PROGRAM ---
2000 REM --- SECOND PROGRAM ---
3000 REM --- THIRD PROGRAM ---

ALL VARIABLES ARE RESET BY THE 'RUN' COMMAND.

2.21 CARTESIAN COORDINATES FROM POLAR EQUATIONS.

100 FOR TH = 0 TO 180 : REM PLOT MANY POINTS
110 R = COS(TH / 57.2958) : REM SAMPLE POLAR FUNCTION
120 GOSUB 200 : REM CONVERT TO (X,Y)
130 NEXT TH : END
140 :
200 REM --- CONVERT RADIUS R AND ANGLE TH TO (X,Y) ---
210 :
220 X = R * SIN(TH / 57.2958)
230 Y = R * COS(TH / 57.2958)
240 :
250 REM --- NOW SCALE AND PLOT COORDINATE (X,Y) ---
260 :
270 RETURN

2.22 SHELL SORT TO PUT ARRAY Z() IN ASCENDING ORDER

100 D = N :REM N=# OF ELEMENTS IN ARRAY Z()
110 D = INT(D/2)
120 FOR I=1 TO N-D
130 IF Z(I) <= Z(I+D) THEN 160
140 T=Z(I) : Z(I)=Z(I+D) : Z(I+D)=T
150 IF I>D THEN IF Z(I-D) > Z(I) THEN I=I-D : GOTO 140
160 NEXT I
170 IF D > 1 THEN 110
180 RETURN
CHAPTER 3 --- FUNCTIONS

3.01 RANDOM NUMBER FUNCTION

\[ J = \text{RND}(X) \] produces the following depending on \( X \):

- \( X > 0 \) \( J \) = A RANDOM NUMBER BETWEEN 0 AND 1.
- \( X = 0 \) \( J \) = THE NUMBER PRODUCED FROM THE LAST RND() AGAIN.
- \( X < 0 \) \( J \) = A RANDOM NUMBER BETWEEN 0 AND 1 WHICH IS ALWAYS THE SAME FOR A GIVEN \( X \). BY SEEDING RND() WITH THE SAME \( X \) A SEQUENCE IS REPEATED.

3.02 RANDOMIZE FUNCTION

100 POKE 318, 237
110 \( X = \text{INP}(95) \)
120 \( J = \text{RND}(-X*2-1) \)

\( X \) = A RANDOM INTEGER BETWEEN 0 AND 127, INCLUSIVE. USE ONLY ODD NEGATIVE NUMBERS TO SEED THE RND() FUNCTION. THE USER HAS NO CONTROL OVER WHICH SEQUENCE IS SELECTED. THROUGHOUT THE REST OF THE PROGRAM USE \( J = \text{RND}(1) \).

3.03 INT FUNCTION EXAMPLES.

\[ \begin{align*}
\text{xx.yy > 0} & \quad \text{INT}(\text{xx.yy}) = \text{xx} & \quad \text{INT}(7.36) = 7 \\
\text{xx.yy < 0} & \quad \text{INT}(\text{xx.yy}) = \text{xx} - 1 & \quad \text{INT}(-7.3) = -8
\end{align*} \]

3.04 SGN() FUNCTION

\[ \text{SGN}(X) = 1 \text{ WHEN } X > 0. \]
\[ \text{SGN}(X) = 0 \text{ WHEN } X = 0. \]
\[ \text{SGN}(X) = -1 \text{ WHEN } X < 0. \]
**3.05 DEF FUNCTION RULES**

The first 2 letters must be 'FN'.
The third character must be a letter.
Additional characters are optional.
The name cannot contain any reserved words.
Only the 2 characters after the FN make the name unique.

Examples: FNA( FNX1( FNAA( FNSQ( FNUPPER(  

The argument variable is a dummy used to 'map' values into the function if desired.
Other variables in the function use their current values.

Example:  
100 DEF FNA(J)= J * J + X  
110 J=5 : X=3 : PRINT FNA(2)  

Output: 7  
Since J was the dummy argument, it mapped a 2 into the function rather than use its current value of 5.  
The current value of X was used. Thus, 7 = 2 * 2 + 3.

**3.06 DECIMAL POINT ALIGNMENT**

100 DEF FNA(J)= (J=0) - (ABS(J)<1) - LEN(STR$(INT(J)))  
110 PRINT TAB(T+FNA(X));X  

All numbers X will be printed with their decimal points aligned in column T.

**3.07 TABULATED PRINTING OF DOLLARS AND CENTS.**

100 DEF FNA(J) = (J=0) - (ABS(J)<1) - LEN(STR$(INT(J)))  
110 FOR I=1 TO 4 : READ X : GOSUB 200 : NEXT I : END  
120 :  
130 DATA 1090, -1090.1, 98.51, 96.3372 : REM 4 EXAMPLES  
140 :  
200 A$ = MID$(STR$(ABS(X) - INT(ABS(X)) + 1.005),4,2)  
210 PRINT TAB(15 + FNA(X));STR$(INT(X));".";A$  
220 RETURN  

Output: 1090.00  
-1090.10  
98.51  
96.34  

Decimal points are aligned.  
Only two decimal digits printed.  
Fractional pennies rounded off.
3.08 ROUND OFF FUNCTION

100 DEF FNA(X) = INT( X * 10 ^ J + 0.5 ) / 10 ^ J

The function FNA rounds off a number to J decimal places.

Example: 110 J=3 : PRINT FNA(2/3)

Output: 0.667

3.09 ASCII CODE FOR A HEX DIGIT

100 DEF FNA(J) = J + 48 - 7*(J>9)

Example: PRINT CHR$(FNA(14))

Output: D

3.10 FIX COMMAND

100 DEF FNA(J) = SGN(J) * INT(ABS(J))

Example: PRINT FNA(-5.2)

Output: -5

3.11 BASE 10 LOGORITHM

100 DEF FNLOG10(J) = LOG(J) * 0.4342945

Example: PRINT FNLOG10(1000)

Output: 3

3.12 PI = 3.14159 = 355/113

57.2958 = 180 / PI

100 DEF FND(J) = J * PI / 180 : REM DEGREES TO RADIANS
110 DEF FNR(J) = J * 180 / PI : REM RADIANS TO DEGREES
CHAPTER 4

4.01 ACCEPT 'YES' ANSWER WITH INPUT OF 'Y', 'YE' OR 'YES'

100 INPUT "ENTER 'YES' OR 'NO'";A$
110 IF LEFT$("YES",LEN(A$))=A$ THEN PRINT "YES"

==============================================================================

4.02 KEYBOARD SCAN OR 'GET' COMMAND

100 POKE 318,195 : POKE 320,224
110 X=INP(9) : A$=CHR$(X)

X = ASCII VALUE OF KEY DEPRESSED. 'RETURN' NOT REQUIRED.
X = 0 IF NO KEY IS DEPRESSED.
A$= CHARACTER KEYED.

==============================================================================

4.03 PRINT ON LINE AFTER INPUT STATEMENT.

100 B$="ENTER EXAMPLE TEXT"
110 PRINT B$;:INPUT A$
120 PRINT TAB(LEN(B$)+LEN(A$)+5);CHR$(23);"THANK YOU"

THE ABOVE EXAMPLE TABS BEYOND THE LENGTH OF THE USER'S
INPUT TO PRINT THE COMPUTER'S RETURN RESPONSE ON THE
SAME LINE AS THE INPUT.

==============================================================================

4.04 INPUT STRINGS CONTAINING COMMAS AND '@' SYMBOLS

100 POKE 318,195 : POKE 320,224
110 GOSUB 200 : PRINT A$ : END
120 :
200 A$="" : REM INPUT A$ UNTIL <CR> RECEIVED
210 J = INP(9) : IF J=13 THEN RETURN
220 A$ = A$ + CHR$(J) : GOTO 210
4.05 KEYBOARD SCAN OF 'GRAPHIC', 'CONTROL', AND 'SHIFT'.

100 J=INP(254) AND 31

J = 31 IF 'CONTROL' KEY IS DEPRESSED.
J = 21 IF 'GRAPHIC' KEY IS DEPRESSED.
J = 7 IF 'SHIFT' KEY IS DEPRESSED.
J = 5 IF 'GRAPHIC - SHIFT' KEYS ARE DEPRESSED.
J = 23 IF 'NO KEY IS DEPRESSED.

'SHIFT LOCK' KEY MUST BE DOWN.

===============================================================

4.06 WAIT FOR USER TO PRESS 'SHIFT' KEY TO CONTINUE.

100 PRINT "PRESS 'SHIFT' TO CONTINUE"
110 WAIT 254,31,23
120 PRINT "I WAS WAITING FOR YOU. THANKS."
CHAPTER 5  --- VIDEO

5.01 DOUBLE SPACE LISTINGS.

POKE 322,0 : LIST

5.02 SCREEN POKE ADDRESS FOR ANY ROW AND COLUMN.

100 DEF PNA(J) = R * 64 + C - 3968

R = ROW NUMBER FROM 0 TO 29.
C = COLUMN NUMBER FROM 0 TO 63.

5.03 MAKE THE CURSOR DISAPPEAR AFTER A PRINT.

100 PRINT CHR$(17); : POKE -3968,32

OR, JUST REPLACE THE CHARACTER UNDER THE CURSOR WITH:

100 POKE 260,232 : POKE 261,233 : J=USR(0)

5.04 REMOVE CURSOR FROM SCREEN.

100 PRINT CHR$(12) : POKE -3904,32

5.05 SCREEN ADDRESSING

ADDRESS OF THE FIRST COLUMN IN EACH VIDEO LINE.

<table>
<thead>
<tr>
<th>LINE#</th>
<th>HEX</th>
<th>POKE DECIMAL</th>
<th>LINE#</th>
<th>HEX</th>
<th>POKE DECIMAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F080</td>
<td>-3968</td>
<td>16</td>
<td>F440</td>
<td>-3008</td>
</tr>
<tr>
<td>2</td>
<td>F0C0</td>
<td>-3904</td>
<td>17</td>
<td>F480</td>
<td>-2944</td>
</tr>
<tr>
<td>3</td>
<td>F100</td>
<td>-3840</td>
<td>18</td>
<td>F4C0</td>
<td>-2880</td>
</tr>
<tr>
<td>4</td>
<td>F140</td>
<td>-3776</td>
<td>19</td>
<td>F500</td>
<td>-2816</td>
</tr>
<tr>
<td>5</td>
<td>F180</td>
<td>-3712</td>
<td>20</td>
<td>F540</td>
<td>-2752</td>
</tr>
<tr>
<td>6</td>
<td>F1C0</td>
<td>-3648</td>
<td>21</td>
<td>F580</td>
<td>-2688</td>
</tr>
<tr>
<td>7</td>
<td>F200</td>
<td>-3584</td>
<td>22</td>
<td>F5C0</td>
<td>-2624</td>
</tr>
<tr>
<td>8</td>
<td>F240</td>
<td>-3520</td>
<td>23</td>
<td>F600</td>
<td>-2560</td>
</tr>
<tr>
<td>9</td>
<td>F280</td>
<td>-3456</td>
<td>24</td>
<td>F640</td>
<td>-2496</td>
</tr>
<tr>
<td>10</td>
<td>F2C0</td>
<td>-3392</td>
<td>25</td>
<td>F680</td>
<td>-2432</td>
</tr>
<tr>
<td>11</td>
<td>F300</td>
<td>-3328</td>
<td>26</td>
<td>F6C0</td>
<td>-2368</td>
</tr>
<tr>
<td>12</td>
<td>F340</td>
<td>-3264</td>
<td>27</td>
<td>F700</td>
<td>-2304</td>
</tr>
<tr>
<td>13</td>
<td>F380</td>
<td>-3200</td>
<td>28</td>
<td>F740</td>
<td>-2240</td>
</tr>
<tr>
<td>14</td>
<td>F3C0</td>
<td>-3136</td>
<td>29</td>
<td>F780</td>
<td>-2176</td>
</tr>
<tr>
<td>15</td>
<td>F400</td>
<td>-3072</td>
<td>30</td>
<td>F7C0</td>
<td>-2112</td>
</tr>
</tbody>
</table>
5.06 SYNC SCREEN MOTION WITH VIDEO HORIZONTAL SYNC.

100 WAIT 254,32

THIS WILL REMOVE SCREEN FLICKER AND DISAPPEARANCE OF CHARACTERS DURING SCREEN MOTION DUE TO BEING OUT OF SYNC WITH THE SCREEN REFRESH CIRCUITRY.

5.07 PRINT AT SUBROUTINE

100 CLEAR 200
110 R$=CHR$(26) : REM CURSOR DOWN
120 C$=CHR$(19) : REM CURSOR RIGHT
130 FOR J=1 TO 6 : R$=R$+R$ : C$=C$+C$ : NEXT J
140 R$=CHR$(17)+R$ : REM ADD CURSOR HOME
150 :
160 REM --- PRINT TEXT AT ROW R, COLUMN C ---
170 :
180 R=5 : C=10 : A$="THIS IS AN EXAMPLE" : GOSUB 1000
190 END
200 :
1000 REM --- PRINT AT SUBROUTINE ---
1010 :
1020 PRINT LEFT$(R$,R);LEFT$(C$,C);A$ : RETURN

5.08 PLACE CURSOR AT ROW AND COLUMN.

THIS ROUTINE MUST BE LOCATED STARTING AT ADDRESS 0, BECAUSE THE 'OUT' INSTRUCTION IS MADE INTO A RST OH COMMAND.

0000: E5 ENTRY PUSH HL
0001: CD A2 E1 CALL 0E1A2H ;GET IY
0004: 2A 07 01 LD HL,(0107H) ;GET ROW #
0007: 26 00 LD H,0
0009: 29 ADD HL,HL ;*2
000A: 29 ADD HL,HL ;*4
000B: 29 ADD HL,HL ;*8
000C: 29 ADD HL,HL ;*16
000D: 29 ADD HL,HL ;*32
000E: 29 ADD HL,HL ;*64
000F: FD 75 68 LD (IY+68H),L ;ROW ADDRESS
0012: FD 74 69 LD (IY+69H),H
0015: FD 77 6A LD (IY+6AH),A ;COL ADDRESS
0018: CD CC E9 CALL 0E9CCH ;MOVE CURSOR
001B: 36 20 LD (HL),20H ;STORE SPACE
001D: E1 POP HL
001E: C1 POP BC ;RELIEVE STACK
001F: C9 RET
CALLING THE ROUTINE WILL MOVE THE CURSOR TO THE REFERENCED SCREEN POSITION AND PLACE A SPACE THERE.

EXAMPLE: 100 POKE 262,199 : REM INSERT RST OH COMMAND
           110 OUT 7,15 : REM CURSOR TO ROW 7, COL 15
           120 PRINT "TEXT STARTS ON ROW 7, COLUMN 15"

================================================================

5.09 PLACE CURSOR AT ROW AND COLUMN.

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000:</td>
<td>ROW</td>
<td>DEFB 0</td>
</tr>
<tr>
<td>0001:</td>
<td>COL</td>
<td>DEFB 0</td>
</tr>
<tr>
<td>0002:</td>
<td>CD A2 E1</td>
<td>ENTRY CALL OE1A2H ;GET IY</td>
</tr>
<tr>
<td>0005:</td>
<td>2A 00 00</td>
<td>LD HL,(ROW)</td>
</tr>
<tr>
<td>0008:</td>
<td>7C</td>
<td>LD A,H</td>
</tr>
<tr>
<td>0009:</td>
<td>26 00</td>
<td>LD H,0</td>
</tr>
<tr>
<td>000B:</td>
<td>29</td>
<td>ADD HL,HL   ;*2</td>
</tr>
<tr>
<td>000C:</td>
<td>29</td>
<td>ADD HL,HL   ;*4</td>
</tr>
<tr>
<td>000D:</td>
<td>29</td>
<td>ADD HL,HL   ;*8</td>
</tr>
<tr>
<td>000E:</td>
<td>29</td>
<td>ADD HL,HL   ;*16</td>
</tr>
<tr>
<td>000F:</td>
<td>29</td>
<td>ADD HL,HL   ;*32</td>
</tr>
<tr>
<td>0100:</td>
<td>29</td>
<td>ADD HL,HL   ;*64</td>
</tr>
<tr>
<td>0111:</td>
<td>FD 75 68</td>
<td>LD (IY+68H),L ;ROW ADDRESS</td>
</tr>
<tr>
<td>0114:</td>
<td>FD 74 69</td>
<td>LD (IY+69H),H</td>
</tr>
<tr>
<td>0117:</td>
<td>FD 77 6A</td>
<td>LD (IY+6AH),A ;COL ADDRESS</td>
</tr>
<tr>
<td>011A:</td>
<td>CD CC E9</td>
<td>CALL 0E9CCH ;MOVE CURSOR</td>
</tr>
<tr>
<td>011D:</td>
<td>36 20</td>
<td>LD (HL),20H ;STORE SPACE</td>
</tr>
<tr>
<td>011F:</td>
<td>C9</td>
<td>RET</td>
</tr>
</tbody>
</table>

POKE ROW NUMBER IN ADDRESS 0. RANGE 0 - 29.
POKE COLUMN NUMBER IN ADDRESS 1. RANGE 0 - 63.

CALLING THE ROUTINE WILL MOVE THE CURSOR TO THE REFERENCED SCREEN POSITION AND PLACE A SPACE THERE.

EXAMPLE: 100 POKE 260,2 : POKE 261,0 : REM USR() ADDR
           110 R=7 : C=15 : GOSUB 200
           120 PRINT "TEXT STARTS ON ROW 7, COLUMN 15"
           130 STOP
           140 :
           200 POKE 0,R : POKE 1,C : J=USR(0) : RETURN
5.10 DRAW BOX ROUTINE

; THIS ROUTINE WILL DRAW A BOX AROUND A TEXT STRING.
; PUT STARTING LOCATING OF STRING IN ROW AND COL, AND ITS
; SIZE IN LENGTH.
; THIS ROUTINE USES THE CURSOR PLACEMENT ROUTINE OF 5.09

CURSOR EQU 02H

ORG 20H
LENGTH DEFB 0

ORG 2430H
BOX LD HL,ROW ;STRING START
DEC (HL)
INC HL
DEC (HL) ;BOX CORNER
LD A,(LENGTH)
LD B,A
CALL CURSOR ;CURSOR TO CORNER
LD (HL),0BCH ;TOP LEFT CORNER
LD DE,040H ;DOWN ONE ROW
ADD HL,DE
LD (HL),0A2H ;LEFT SIDE
ADD HL,DE ;DOWN ONE ROW
LD (HL),0BEH ;BOTTOM CORNER
PUSH HL ;SAVE CORNER ADDR
LD DE,OFF80H ;UP THREE ROWS
ADD HL,DE
POP DE
INC HL ;HL = TOP ADDR
INC DE ;DE = BOTTOM ADDR
LD (HL),097H ;DRAW TOP LINE
EX DE,HL
LD (HL),097H ;DRAW BOTTOM LINE
EX DE,HL
DJNZ HORZ-$ ;LOOP ON LENGTH
LD DE,040H
INC HL
LD (HL),0BDH ;TOP RIGHT CORNER
ADD HL,DE
LD (HL),0A2H ;RIGHT SIDE
ADD HL,DE
LD (HL),0BFH ;BOTTOM CORNER
LD A,000H
LD L,000H
JP CURSOR+7 ;CURSOR TO HOME
5.11 CREATE INVERSE VIDEO CHARACTER SET

100 FOR J=-1024 TO -1
110 POKE J,256 + NOT PEEK(J-1024)
120 NEXT J

USE THE 'GRAPHIC' AND THE 'SHIFT GRAPHIC' KEYS TO DISPLAY THE INVERSE VIDEO CHARACTER SET.

THE SAME THING IS DONE BELOW IN MACHINE LANGUAGE:

0000: E5 ENTRY PUSH HL ;SAVE REGS
0001: D5 PUSH DE
0002: C5 PUSH BC
0003: F5 PUSH AF
0004: 21 00 F8 LD HL,0F800H ;CHARACTER SOURCE
0007: 11 00 FC LD DE,0FC00 ;DESTINATION
000A: 01 00 04 LD BE,0400H ;1024 COUNTER
000D: 7E LOOP LD A,(HL) ;GET A ROW
000E: 2F CPL ;INVERT IT
000F: 12 LD (DE),A ;MOVE IT DOWN
0010: 23 INC HL ;HL = HL + 1
0011: 13 INC DE ;DE = DE + 1
0012: 0B DEC BC ;COUNT IT
0013: 20 F8 JR NZ,LOOP-$ ;REPEAT TIL DONE
0015: F1 POP AF
0016: C1 POP BC
0017: D1 POP DE
0018: E1 POP HL ;RESTORE REGS
0019: C9 RET

NOW THAT THE INVERSE CHARACTER SET HAS BEEN CREATED, LET'S PRINT INVERSE TEXT FROM THE STRING A$.

200 INPUT A$ : GOSUB 300 : GOTO 200
300 :
310 REM THIS ROUTINE PRINTS A$ IN INVERSE VIDEO
320 :
330 FOR J=1 TO LEN(A$)
340 PRINT CHR$(ASC(MID$(A$,J)) OR 128) ; NEXT J : RETURN
5.12 CREATE DOUBLE WIDE CHARACTER SET

64 ASCII CHARACTERS FROM ASCII 20H (SPACE) THROUGH ASCII 5FH (UNDERSCORE) WILL BE MADE INTO DOUBLE WIDE CHARACTERS USING THE 128 AVAILABLE GRAPHIC CHARACTERS. THIS DOUBLE WIDE CHARACTER SET INCLUDES THE DIGITS, SYMBOLS, AND THE UPPER CASE LETTERS.

THIS ROUTINE WILL CREATE THE DOUBLE WIDE CHARACTER SET.

```
100 FOR J= 32 TO 95 : I=(J-256)*8 : K=(2*J-192)*8
110 FOR L= 0 TO 7 : M=PEEK(I+L) : N=INT(M/16) : P=M AND 15
120 R=O : S=O : FOR T= 0 TO 3 : V=2^T
130 R=R+(N AND V)*V*3 : S=S+(P AND V)*V*3 : NEXT T
140 POKE K+L,R : POKE K+L+8,S : NEXT L,J
```

A CHARACTER WITH ASCII VALUE J CAN BE PRINTED IN DOUBLE WIDTH USING THE TWO GRAPHIC CHARACTERS WITH ASCII VALUE OF 2*J + 64 AND 2*J + 65. SEE THE FOLLOWING EXAMPLE.

```
150 A$="SAMPLE TEXT STRING TO PRINT"
160 FOR I=1 TO LEN(A$) : J=ASC(MID$(A$,I,1)+64
170 PRINT CHR$(J);CHR$(J+1); : NEXT I
```

OR, YOU MAY WISH TO HAVE THE CHARACTERS SENT TO THE VIDEO DISPLAYED IN DOUBLE SIZE AUTOMATICALLY. THIS CAN BE ACCOMPLISHED USING THE FOLLOWING VIDEO DRIVER. CHANGE THE >SE 0=xxxy VECTOR ADDRESS TO POINT TO THIS ROUTINE, IE. >SE 0=0000, OR POKE 32720,0 POKE 32721,0 FOR 32K.

```
0000: FE 20 CP 32 ;CHAR BEFORE 'SPACE'
0002: FA 1B E0 JP M,VIDEO ;YES, EXIT TO VIDEO
0005: 87 ADD A,A ;J*2
0006: C6 40 ADD A,64 ;J*2+64
0008: CD 1B E0 CALL VIDEO ;PRINT LEFT HALF
000B: 3C INC A ;J*2+65
000C: C3 1B E0 JP VIDEO ;PRINT RIGHT HALF
```
CHAPTER 6 --- JOYSTICKS

6.01 JOYSTICK / KEYBOARD STANDARD FOR THE SORCERER

This standard has been adopted by several software houses in the USA and in Australia. Software offered by these vendors which employs joystick/keyboard control will conform to this standard. It is suggested that all SORCERER owners use this standard for international compatibility of software and hardware.

Two joysticks may be attached to the INPUT of the parallel port. UNIT #1 uses the LOW-order 4 bits, and UNIT #2 uses the HIGH-order 4 bits. Each unit may steer in the four basic directions, LEFT, RIGHT, UP, DOWN, as well as in the four diagonal directions. Both units operate independently, and simultaneous operation is permitted.

FIRE BUTTON control may be included, and has priority over directional control of the joystick unit it is attached to. FIRE BUTTON is activated by grounding both BIT 0 and BIT 1 for unit #1, and BIT 4 and BIT 5 for unit #2.

KEYBOARD has priority over JOYSTICK, and overrides both joystick units if used. KEYBOARD INPUT RESULT is returned as the RESULT CODE of joystick unit #1, with joystick unit #2 disabled.

Keyboard directional control is via the "arrow" (normally cursor control) keys in the NUMERIC KEYPAD only. The SHIFT key need not be depressed when using these keys. FIRE BUTTON on the keyboard is the NUMERIC-PAD "5" key (HOME). Optional FIRE BUTTONS may be SKIP/TAB or SPACE BAR. FIRE BUTTON overrides directional keys on the keyboard.

In the event that both the LEFT and the RIGHT keys are pressed together, it is treated as NO INPUT. The same rule applies to depressing both the UP and the DOWN keys together. The UP/LEFT ("7"), UP/RIGHT ("9"), DOWN/LEFT ("1") and DOWN/RIGHT ("3") keys on the numeric-pad are optional.

For programming in Z80 machine code, the 8-bit INPUT RESULT CODE is returned in the A-register. No other registers are affected. If there is no input, then return with a zero in the A-register and with the Z-flag set.
6.02 JOYSTICK INTERFACE STANDARD TO PARALLEL PORT

<table>
<thead>
<tr>
<th>BIT</th>
<th>PIN</th>
<th>FUNCTION</th>
<th>BIT</th>
<th>PIN</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>UNIT #1 LEFT</td>
<td>4</td>
<td>12</td>
<td>UNIT #2 LEFT</td>
</tr>
<tr>
<td>1</td>
<td>22</td>
<td>UNIT #1 RIGHT</td>
<td>5</td>
<td>24</td>
<td>UNIT #2 RIGHT</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>UNIT #1 UP</td>
<td>6</td>
<td>13</td>
<td>UNIT #2 UP</td>
</tr>
<tr>
<td>3</td>
<td>23</td>
<td>UNIT #1 DOWN</td>
<td>7</td>
<td>25</td>
<td>UNIT #2 DOWN</td>
</tr>
<tr>
<td>0-1</td>
<td>10/22</td>
<td>UNIT #1 FIRE</td>
<td>4-5</td>
<td>12/24</td>
<td>UNIT #2 FIRE</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Ground</td>
<td>20</td>
<td></td>
<td>+5 VOLT SUPPLY</td>
</tr>
</tbody>
</table>

6.03 JOYSTICK CIRCUIT DIAGRAM

ATARI joysticks can be easily modified to connect directly to the parallel port. A 4.7K 1/4 watt resistor pulls-up each direction input to +5 volts at the nodes marked with 'Y' in the diagram. The FIRE BUTTON employs the two diodes between the LEFT and the RIGHT direction inputs, and GROUND connects to the common line. When the joystick selects a direction, or the fire button is pressed, a switch closes which changes the input bit from +5 volts to ground.
6.04 JOYSTICK EXAMPLE USING BASIC STATEMENTS

100 A = 255 - INP(255) : REM READ PARALLEL PORT
110 IF (A AND 3)= 3 THEN "FIRE BUTTON UNIT #1"
120 IF (A AND 48)=48 THEN "FIRE BUTTON UNIT #2"

130 IF A AND 1 THEN "UNIT #1 LEFT"
140 IF A AND 2 THEN "UNIT #1 RIGHT"
150 IF A AND 4 THEN "UNIT #1 UP"
160 IF A AND 8 THEN "UNIT #1 DOWN"

170 IF A AND 16 THEN "UNIT #2 LEFT"
180 IF A AND 32 THEN "UNIT #2 RIGHT"
190 IF A AND 64 THEN "UNIT #2 UP"
200 IF A AND 128 THEN "UNIT #2 DOWN"

210 GOTO 100 : REM SCAN AGAIN

Use the Basic statements to observe the state of the joysticks. The text strings tell what to do after determining the condition of the joysticks. If you branch to a routine to service Unit #1, be sure to return to the testing of Unit #2 so that it can operate simultaneous with Unit #1.

=================================================================

6.05 JOYSTICK EXAMPLE USING MACHINE LANGUAGE CODE

The joystick source listing of section 6.06 is a useful routine that loads into memory from address 0 through A4 hex. To use the routine from a Basic program, set the USR() jump address to 0 as shown on line 100 below.

Now initialize the location of each joystick cursor by placing a screen row number and a column number in the following addresses using POKE statements: (See example on line 110 below.)

ADDRESS 2 - ROW # UNIT #1, RANGE (1...30).
3 - COLUMN # UNIT #1, RANGE (1...64).
4 - ROW # UNIT #2, RANGE (1...30).
5 - COLUMN # UNIT #2, RANGE (1...64).
6 - FIRE FOR BOTH UNITS: = 1 UNIT #1 ONLY
   = 2 UNIT #2 ONLY
   = 3 BOTH UNITS FIRING.
7,8 - SCREEN ADDRESS OF UNIT #1
9,10 - SCREEN ADDRESS OF UNIT #2

100 POKE 260,0:POKE 261,0:PRINT CHR$(12):REM USR() ADDR
110 POKE 2,1:POKE 3,1:POKE 4,1:POKE 5,1 :REM INITIALIZE
120 Z=USR(0) :REM GET JOYSTICK
130 L=PEEK(2)*64+PEEK(3)-4033 :REM SCREEN ADDR
140 POKE L,ASC("1") :REM CURSOR #1
150 M=PEEK(9)+PEEK(10)*256-65536 :REM ANOTHER WAY
160 POKE M,50 :REM CURSOR #2
170 GOTO 120 :REM DO AGAIN
6.06 MACHINE LANGUAGE JOYSTICK ROUTINE

This routine reads the status of the joysticks connected to the parallel port and updates the ROW, COLUMN and FIRE variables for each joystick. The routine will keep the row variables in the range of (1...30), and the column variables in the range of (1...64). The user's Basic program can access these variables to compute the screen position of each joystick, or read directly the contents of ADDR1 and ADDR2. Section 6.05 gives an example of how Basic might access this routine.

```
0000: 18 09 ENTRY JR START-$

0002: 01 ROW1 DEFB 1
0003: 01 COL1 DEFB 1
0004: 01 ROW2 DEFB 1
0005: 01 COL2 DEFB 1
0006: 00 FIRE DEFB 0
0007: 00 00 ADDR1 DEFW 0
0009: 00 00 ADDR2 DEFW 0

; LEFT1 EQU 0
RIGHT1 EQU 1
UP1 EQU 2
DOWN1 EQU 3
FIRE1 EQU 3
LEFT2 EQU 4
RIGHT2 EQU 5
UP2 EQU 6
DOWN2 EQU 7
FIRE2 EQU 48

000B: DB FF START IN A,(OFFH) ;GET JOYSTICK
000D: 2F CPL
000E: B7 OR A ;ANY ACTIVITY
000F: 21 06 00 LD HL,FIRE
0012: 36 00 LD (HL),000H ;REMOVE FIRES
0014: C8 RET Z ;RET IF IDLE

0015: F5 PUSH AF
0016: E6 03 AND FIRE1
0018: FE 03 CP FIRE1
001A: 20 01 JR NZ,L001D-$
001C: 34 INC (HL) ;FIRE1 ACTIVE
001D: F1 L001D POP AF

001E: F5 PUSH AF
001F: E6 30 AND FIRE2
0021: FE 30 CP FIRE2
0023: 20 02 JR NZ,L0027-$
0025: 34 INC (HL) ;FIRE2 ACTIVE
0026: 34 INC (HL)
```
: H = COLUMN 1  L = ROW 1
: D = COLUMN 2  E = ROW 2
: INCREMENT COL IF RIGHT
: INCREMENT ROW IF DOWN
: DECREMENT COL IF LEFT
: DECREMENT ROW IF UP

0027: F1 L0027 POP AF
0028: 2A 02 00 LD HL,(ROW1)
002B: ED 5B 04 00 LD DE,(ROW2)
002F: CB 47 BIT LEFT1,A
0031: 28 01 JR Z,L0034-$
0033: 25 DEC H
0034: CB 67 L0034 BIT LEFT2,A
0036: 28 01 JR Z,L0039-$
0038: 15 DEC D
0039: CB 4F L0039 BIT RIGHT1,A
003B: 28 01 JR Z,L003E-$
003D: 24 INC H
003E: CB 6F L003E BIT RIGHT2,A
0040: 28 01 JR Z,L0043-$
0042: 14 INC D
0043: CB 57 L0043 BIT UP1,A
0045: 28 01 JR Z,L0048-$
0047: 2D DEC L
0048: CB 77 L0048 BIT UP2,A
004A: 28 01 JR Z,L004D-$
004C: 1D DEC E
004D: CB 5F L004D BIT DOWN1,A
004F: 28 01 JR Z,L0052-$
0051: 2C INC L
0052: CB 7F L0052 BIT DOWN2,A
0054: 28 01 JR Z,L0057-$
0056: 1C INC E

: COLUMN RANGE: 1 - 64
: ROW RANGE: 1 - 30

0057: 3E 1F L0057 LD A,31 ;ROW LIMIT
0059: BD CP L
005A: 20 01 JR NZ,L005D-$
005C: 2D DEC L
005D: BB L005D CP E
005E: 20 01 JR NZ,L0061-$
0060: 1D DEC E
0061: 3E 41 L0061 LD A,65 ;COL LIMIT
0063: BC CP H
0064: 20 01 JR NZ,L0067-$
0066: 25 DEC H
0067: BA L0067 CP D
0068: 20 01 JR NZ,L006B-$
006A: 15 DEC D
; CONVERT ROW AND COL TO A SCREEN ADDRESS STORED IN ADDR1 AND ADDR2
; ADDR = SCREEN BASE + ROW * 64 + COL

006B: 3E 00        L006B LD A,000H ;KEEP > 0
006D: BD          CP L
006E: 20 01        JR NZ,L0071-$
0070: 2C          INC L
0071: BB          L0071 CP E
0072: 20 01        JR NZ,L0075-$
0074: 1C          INC E
0075: BC          L0075 CP H
0076: 20 01        JR NZ,L0079-$
0078: 24          INC H
0079: BA          L0079 CP D
007A: 20 01        JR NZ,L007D-$
007C: 14          INC D
007D: 22 02 00    L007D LD (ROW1),HL ;ROW1 - COL1
0080: ED 53 04 00  LD (ROW2),DE ;ROW2 - COL2
0084: CD 94 00     CALL ADDRESS
0087: 22 07 00     LD (ADDR1),HL ;SCREEN ADDR1
008A: 2A 04 00     LD HL,(ROW2)
008D: CD 94 00     CALL ADDRESS
0090: 22 09 00     LD (ADDR2),HL ;SCREEN ADDR2
0093: C9          RET

0094: 5C          ADDRESS LD E,H ;DE = COL
0095: 16 00       LD D,000H
0097: 26 00       LD H,000H ;HL = ROW
0099: 29          ADD HL,HL ;*2
009A: 29          ADD HL,HL ;*4
009B: 29          ADD HL,HL ;*8
009C: 29          ADD HL,HL ;*16
009D: 29          ADD HL,HL ;*32
009E: 29          ADD HL,HL ;HL = ROW*64
009F: 19          ADD HL,DE ;ROW + COL
00A0: 11 3F F0     LD DE,0F03FH ;SCREEN BASE
00A3: 19          ADD HL,DE
00A4: C9          RET
CHAPTER 7 --- SOUND

7.01 PRINCIPLE OF ONE VOICE SOUND

Generating sound from your Sorcerer is not that difficult, and this discussion will help you get started by detailing a machine language routine to generate music.

Let's use the parallel port as a means to output a signal to an external speaker since the Sorcerer does not have an internal speaker like some other computers do. I DO NOT recommend connecting a small 8 ohm speaker directly between one of the parallel port's output bits and ground. It is safer to have an output bit drive a transistor's base via a 1K resistor, and let the transistor switch current through a small speaker. Or, you may wish to use the music interface board that comes with our four voice Music System.

The statement: 10 FOR I=1 TO 100:OUT 255,255:OUT 255,0:NEXT I generates a low pitched note by toggling the output bits on the parallel port from high to low, back to high. It may be sufficient for your needs, but provides no control over the pitch of the sound. However, it illustrates the principle of how sound will be generated in machine language.

The following code generates notes.

```
0000: F5 ENTER PUSHPUSH AF ;SAVE REGISTERS USED
0001: C5 PUSHB
0002: E5 PUSHP
0003: 21 60 00 LDHL,DURATION ;LENGTH OF SOUND
0006: 79 TOP LD A,C
0007: 2F CPL ;TOGGLE OUTPUT BITS
0008: 4F LD C,A
0009: 06 40 LD B,PITCH ;FREQUENCY CONTROL
000B: D3 FF OUT (OFFH),A ;TO PARALLEL PORT
000D: 10 FE LOOP DJNZ LOOP-$ ;DELAY
000F: 2B DEC HL ;DOWN COUNT DURATION
0010: 7D LD A,L
0011: B4 OR H ;IS COUNT = 0 ?
0012: 20 F2 JR NZ,TOP-$ ;LOOP WHILE HL<>0
0014: E1 POP HL
0015: C1 POPBC
0016: F1 POPAF ;RESTORE REGISTERS
0017: C9 RET ;RETURN FROM SUBROUTINE
```

You can vary the pitch by changing the values loaded into register B (ie. contents of address OAH.) How long the note is played is controlled by the duration value loaded into HL (ie. the contents of address O4H and O5H.)

The routine above suffers from the effect of having the delay loop nested inside of the duration loop. If the delay loop is tight, then the duration is accordingly shortened, since the total time in the routine is the product of the two loop parameters. A routine which has a duration independent of the frequency is given in 7.02
7.02 ONE VOICE MUSIC ROUTINE

FREQ EQU OFCH ;STORE PITCH # IN 253
DURATION EQU OFDH ;DURATION # IN 254 - 255.
SPEED EQU 14FFH ;TEMPO FOR ENTIRE SONG.
                 ;
                 ;ORG ODOH ;LOCATE ON ZERO PAGE
 00D0: F5 PUSH AF ;SAVE REGISTERS
 00D1: C5 PUSH BC
 00D2: D5 PUSH DE
 00D3: E5 PUSH HL
 00D4: FD E5 PUSH IY
 00D6: 21 FC 00 ENTER LD HL,FREQ
 00D9: 4E LD C,(HL) ;GET FREQUENCY #
 00DA: 11 FF 14 LD DE,SPEED
 00DD: FD 2A FD 00 LD IY,(DUR) ;GET DURATION
 00E1: D3 FF LOOP OUT (OFFH),A ;TO PARALLEL PORT
 00E3: 0D DEC C ;IS IT TIME TO TOGGLE
 00E4: C2 E9 00 JP NZ,SKIP
 00E7: 2F CPL ;TOGGLE OUTPUT
 00E8: 4E LD C,(HL) ;RELOAD FREQUENCY COUNT
 00E9: FD 19 SKIP ADD IY,DE
 00EB: DA E1 00 JP C,LOOP ;LOOP UNTIL DURATION UP
 00EE: FD E1 POP IY
 00F0: E1 POP HL
 00F1: D1 POP DE
 00F2: C1 POP BC
 00F3: F1 POP AF ;RESTORE REGISTERS
 00F4: C9 RET ;RETURN

To use the above routine from Basic, store the frequency number in byte 253 with POKE 253,FREQ. Store the duration in bytes 254, and 255 with POKE 254,DURATION:POKE 255,0. Usually control from (254) is sufficient, and keep (255) at zero.

Call the routine through the USR() function by poking the entry address into 260, and 261: POKE 260,214:POKE 261,0:X=USR(0).

Here are frequency numbers to generate notes for one octave. Higher octaves can be obtained by dividing these numbers by 2, 4, and 8, etc.

<table>
<thead>
<tr>
<th>NOTE</th>
<th>FREQUENCY</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>268</td>
</tr>
<tr>
<td>C#</td>
<td>253</td>
</tr>
<tr>
<td>D</td>
<td>239</td>
</tr>
<tr>
<td>D#</td>
<td>226</td>
</tr>
<tr>
<td>E</td>
<td>213</td>
</tr>
<tr>
<td>F</td>
<td>201</td>
</tr>
<tr>
<td>F#</td>
<td>190</td>
</tr>
<tr>
<td>G</td>
<td>179</td>
</tr>
<tr>
<td>G#</td>
<td>169</td>
</tr>
<tr>
<td>A</td>
<td>160</td>
</tr>
<tr>
<td>A#</td>
<td>151</td>
</tr>
<tr>
<td>B</td>
<td>142</td>
</tr>
</tbody>
</table>

Although the above music routine is fun and easy, its usefulness is no comparison to the enjoyment you'll get from the four voice Music System from Arrington Software Service.
7.02 ONE VOICE MUSIC ROUTINE

```
FREQ EQU OFCH ;STORE PITCH # IN 252
DURATION EQU OFDH ;DURATION # IN 254 - 253.
SPEED EQU FFFFH ;TEMPO FOR ENTIRE SONG.

ORG ODOH ;LOCATE ON ZERO PAGE

00D0: P5 PUSH AF ;SAVE REGISTERS
00D1: C5 PUSH BC
00D2: D5 PUSH DE
00D3: E5 PUSH HL
00D4: F5 E5 PUSH IY
00D6: 21 FC 00 ENTER LD HL,FREQ
00D9: 4E LD C,(HL) ;GET FREQUENCY #
00DA: 11 FF FF LD DE,SPEED
00DD: FD 2A FD 00 LD IY,(DUR) ;GET DURATION
00E1: D3 FF LOOP OUT (OFFH),A ;TO PARALLEL PORT
00E3: 0D DEC C ;IS IT TIME TO TOGGLE
00E4: C2 E9 00 JP NZ,SKIP
00E7: 2F CPL ;TOGGLE OUTPUT
00E8: 4E LD C,(HL) ;RELOAD FREQUENCY COUNT
00E9: FD 19 SKIPIY,DE ADD IY,DE
00EB: DA E1 00 JP C,LOOP ;LOOP UNTIL DURATION UP
00EE: FD E1 POP IY
00F0: E1 POP HL
00F1: D1 POP DE
00F2: C1 POP BC
00F3: F1 POP AF ;RESTORE REGISTERS
00F4: C9 RET ;RETURN
```

To use the above routine from Basic, store the frequency number in byte 252 with POKE 252,FREQ. Store the duration in bytes 254, and 253 with POKE 254,DURATION:POKE 253,0. Usually control from (254) is sufficient, and keep (253) at zero.

Call the routine through the USR() function by poking the entry address into 260, and 261: POKE 260,208:POKE 261,0:X=USR(0).

Here are frequency numbers to generate notes for one octave. Higher octaves can be obtained by dividing these numbers by 2, 4, and 8, etc.

```
C - 268  C# - 253  D - 239  D# - 226
E - 213  F - 201  F# - 190  G - 179
G# - 169  A - 160  A# - 151  B - 142
```

Although the above music routine is fun and easy, its usefulness is no comparison to the enjoyment you'll get from the four voice Music System from Arrington Software Service.
7.03 PIEZO SPEAKER AUDIO PROMPTER

A miniature 4.7 KHz piezo speaker can be purchased at Radio Shack for three dollars and connected directly to the Sorcerer's parallel port. Connect the red wire to pin 4, and the black wire to pin 8.

The piezo speaker is a self contained chamber which resonates at a fixed frequency when activated with a voltage between 3 and 9 volts. Although I chose to connect the red wire to bit 7 of the port, any of the eight bit outputs will work equally well. The statement of OUT 255,128 causes bit 7 of the port to go to a logical high, thus sourcing the speaker with 5 volts. The 5 volts causes the speaker to continually sound until bit 7 is returned to a logical low with an OUT 255,0. The 255 is the address of the parallel port.

Sending the command sequence of OUT 255,128:OUT 255,0 will cause the speaker to chirp since it sees a brief 5 volt pulse. This audio addition to the Sorcerer is inexpensive and has found frequent use in programs to signal an event in a program's execution, or to prompt the user that an input is required at the keyboard.

This speaker cannot be used to reproduce the music spoken of in 7.01, 7.02, and 7.04. Also, it will not produce the sound that accompanies much of the game software on the market.

=================================================================

7.04 CONTROL 'G' BEEP

THIS ROUTINE INTERCEPTS CONTROL 'G' CHARACTERS SENT TO THE VIDEO DRIVER AND OUTPUTS A BEEP. USE THE MUSIC INTERFACE BOARD FROM OUR MUSIC SYSTEM.

ACTIVATE WITH >SE 0=0

0000: FE 07 ENTER CP 7 ;CONTROL G ?
0002: C2 F0 E9 JP NZ,VIDEO ;EXIT IF NOT
0005: C5 PUSH BC
0006: E5 PUSH HL
0007: 21 60 00 LD HL,DURATION ;LENGTH OF
000A: 79 TOP LD A,C
000B: 2F CPL ;TOGGLE OUTPUT
000C: 4F LD C,A
000D: 06 40 LD B,PITCH ;FREQUENCY
000F: D3 FF OUT (OFFH),A ;PARALLEL PORT
0011: 10 FE LOOP DJNZ LOOP-$$ ;DELAY
0013: 2B DEC HL ;DOWN COUNT
0014: 7D LD A,L
0015: B4 OR H ;IS COUNT=0 ?
0016: 20 F2 JR NZ,TOP-$$ ;LOOP IF HL<>0
0018: E1 POP HL
0019: C1 POP BC
001A: C9 RET ;RETURN
**CHAPTER 8 --- BASIC ROMPAC**

**8.01 BASIC ROMPAC MAP by P. HOLMICK**

**NOTES:**
1. Floating Point variables occupy 4 bytes.
2. ACC is the floating point accumulator 01BF to 01C2.
   - (01BF) = Least significant byte.
   - (01C0) = Next most significant byte.
   - (01C1) = Most significant byte.
   - (01C2) = Exponent.
3. NTF is the Number Type Flag (0190).
   - (0190) = 1 for Strings
   - (0190) = 0 for Numbers
4. For arithmetic operations requiring 2 operands, one is usually the ACC, and the other is either (HL) or BCDE, where BCDE contains:

   - B = Exponent.
   - C = Most significant byte.
   - D = Next most significant byte.
   - E = Least significant byte.
5. HL is used as the text pointer for most of the comparison or conversion routines.

**ADDRESS**

**FUNCTIONAL DESCRIPTION**

- **C000:** Moves a block of memory from C258-C2A6 to 0100-014E.
- **C06B:** BASIC Warm Start
- **C075:** "BYTES FREE" string.
- **C080:** String address for double CR-LF.
- **C082:** String address for single CR-LF.
- **C085:** "EXIDY STANDARD BASIC VER 1.0 .....etc." message string.
- **C0C6:** Command Jump Table.

```
D606,D6CA,D61C,0103,CEE9,D24A,CF17,D8BA,D999,D4AB,D908,
DA0E,DA14,DA75,DA8A,D6FE,D18B,CF9F,D225,D19A,D1AB,D1BB,
D1EB,D1F5
```

- **COF6:** BASIC Token Table.

```
END, FOR, NEXT, DATA, BYE, INPUT, DIM, READ, LET, GOTO, RUN, IF,
RESTORE GOSUB, RETURN, REM, STOP, OUT, ON, NULL, WAIT, DEF, POKE,
PRINT, CONT, LIST, CLEAR, CLOAD, CSAVE, NEW, TAB(.), TO, FN, SPC(.,
THEN, NOT, STEP, +, -, *, /, ^, AND, OR, >, =, <, SGN, INT, ABS, USR,
FRE, INP, POS, SQR, RND, LOG, EXP, COS, SIN, TAN, ATN, PEEK, LEN,
STR$, VAL, ASC, CHR$, LEFT$, RIGHT$, MID$
```

- **C1E1:** Command Jump Table continued.

```
C709,C62E,CB34,C8B5,E003,CA43,CD4F,CA72,C8CC,C872,C855,
C944,C6DD,C861,C890,C8B7,C707,D256,C926,C748,D25C,CF1F,
D705,C968,C735,C5C6,C80F,D341,D2C9,C41A
```
C21D: Jump Table for Arithmetic Operators (+,-,*,/,^,AND,OR)
D7A2,D3AA,D4EA,D54B,D8C3,CCA8,CCA7
Includes a one-byte "precedence" value for each operator which tells the Interpreter the order of arithmetic computations.

C232: Start of Error Codes Table (2 Bytes each)
C258: Start of 4EH bytes to be copied into BASIC CONTROL AREA from 0100H
C2A3: "ERROR" message
C2AA: "IN" message.
C2AF: "READY<CR><LF>" message.
C2B7: "BREAK" message.
C2BD: Used by <FOR>...<NEXT> loops to manipulate SP so that <NEXT> knows where the last <FOR> statement ends.
C2E0: Gets variable name from a line of BASIC - loads it into the variable area.
C2E6: Move bottom of program further down in memory.
BC = Destination address.
DE = Stopping address, ie. stop when HL = DE.
HL = End of program address, ie. source address.

C2F1: Check to see that there's enough free memory for next operation.
C309: OM ERROR
C30E: Puts last-used <DATA> line number into (0147) so that BASIC will print "?SN ERROR IN xx". Where xx=Data Line Number
C314: SN ERROR
C317: /0 ERROR
C31A: NF ERROR
C320: UF ERROR
C322: Prints error message. To use this routine, load the E register with 0 to 24H before jumping here.

===============================
Error Message E = nn
===============================

<table>
<thead>
<tr>
<th>Error</th>
<th>Message</th>
<th>E = 0n</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF</td>
<td>Next Without For</td>
<td>00</td>
</tr>
<tr>
<td>SN</td>
<td>Syntax</td>
<td>02</td>
</tr>
<tr>
<td>RG</td>
<td>Return Without Gosub</td>
<td>04</td>
</tr>
<tr>
<td>OD</td>
<td>Out of Data</td>
<td>06</td>
</tr>
<tr>
<td>FC</td>
<td>Illegal Function Call</td>
<td>08</td>
</tr>
<tr>
<td>OV</td>
<td>Calculation Overflow</td>
<td>0A</td>
</tr>
<tr>
<td>OM</td>
<td>Out of Memory</td>
<td>0C</td>
</tr>
<tr>
<td>UL</td>
<td>Undefined Line Number</td>
<td>0E</td>
</tr>
<tr>
<td>BS</td>
<td>Bad Subscript</td>
<td>10</td>
</tr>
<tr>
<td>DD</td>
<td>Redimensioned Array</td>
<td>12</td>
</tr>
<tr>
<td>/0</td>
<td>Division by Zero</td>
<td>14</td>
</tr>
<tr>
<td>ID</td>
<td>Illegal Direct Mode</td>
<td>16</td>
</tr>
<tr>
<td>TM</td>
<td>Type Mismatch</td>
<td>18</td>
</tr>
<tr>
<td>OS</td>
<td>Out of String Space</td>
<td>1A</td>
</tr>
<tr>
<td>LS</td>
<td>String too Long</td>
<td>1C</td>
</tr>
<tr>
<td>ST</td>
<td>String Formula too complex</td>
<td>1E</td>
</tr>
<tr>
<td>CN</td>
<td>Can't Continue</td>
<td>20</td>
</tr>
<tr>
<td>UF</td>
<td>Undefined User Function</td>
<td>22</td>
</tr>
<tr>
<td>MO</td>
<td>Missing Operand</td>
<td>24</td>
</tr>
</tbody>
</table>
C359: Print "READY" - back to DIRECT COMMAND mode.
C366: Same as C359 except doesn't print "READY".
C3DD: Reset Program pointers - rejustify Link Pointers.
C3E0: Rejustify Link Pointers (see also D394). This can be used to RECOVER a BASIC program after <NEW> or RESET or even <CLOAD> if the original program hasn't been over-written.
C3FA: Searches for a particular Line Number in a BASIC program.
To search: DE = required line number
If found: BC = Start of Line
          HL = Start of next line.
          Carry Flag is set (=1).

C41A: <NEW>
C426: Part of <NEW> routine. Entering here resets HIMEM, STRING Space, and variable area pointers BUT doesn't destroy the first Link bytes thus not "erasing" the program.
C45A: Keyboard Input. Prints "?" then jumps to C53A.
C467: This routine scans the input Buffer and converts lower case to upper case (unless the text is enclosed in quotes) - converts BASIC reserved words to TOKENs.
C51A: Resets HL = 014B and puts 3 zeros at end of program.
C53A: Inputs a string of characters from the keyboard until RETURN is pressed. Uses the buffer from 014C to 018C. On exit, HL=014B and B=number of characters entered.
C555: This is where BASIC tests for buffer overflow but DOESN'T do anything about it !!! (Perhaps JR Z C550 would be a suitable patch.)
C574: Compares HL with DE and sets the flags accordingly.
Z flag set when DE = HL.
C flag set when DE < HL.
C57A: Tests for "expected" characters such as commas, semi-colons, and left and right brackets.
To use: HL should point to the BASIC text to test.
CALL C57A
"", " Test for comma.

NOTE: If a match is NOT made, then an ?SN error results !

C585: Prints the character in the A register. This calls the Monitor Send Vector, E00C.
C5B4: This could be used to implement a <GET> or <INKEY$> function. Inputs one character to the A register. Calls the Monitor Receive Vector, E009, but will not return until a key is pressed !
C5C6: <LIST>
C62E: <FOR>
C66E: <STEP>
C69: Re-entry point to BASIC interpreter. HL should be pointing to the zero byte at the end of the line, or the colon in a multi-statement line.
C6AD: BASIC Interpreter starts here. HL should point to the byte before the actual text to be interpreted.
C6CD: Skips over blanks in the BASIC text by incrementing HL. The Carry Flag is set if the next character is a number.
C6DD: <RESTORE>
C6F8: This is the routine that's supposed to pause LISTings if ESC or RUN/STOP was pressed, however the byte at C701 is wrong. It should be 1BH instead of 13H.
C707: <STOP>
C709: <END>
C70E: Prints "BREAK" if CONTROL-C pressed in DIRECT MODE, else "BREAK IN ....line number" if in RUN MODE.
C735: <CONT>
C748: <NULL>
C75A: CLOAD* jumps here. (01AF) = FF and (01AC) = 01
C75F: CSAVE* jumps here. (01AF) = 01 and (01AC) = 01
C789: Write 4 bytes to tape. Used by CSAVE*
C792: Input 4 bytes from tape. Used by CLOAD*
C7B6: Turn off cassette motors - finish up.
C7BC: Tests for alpha-character pointed to by HL, and resets Carry Flag if found.
C7C4: As for C7D0 below, but first evaluates a BASIC expression pointed to by HL, then puts result into ACC.
C7D0: DE = INTeger value of floating point value in ACCumulator
C7E5: FC ERROR
C7EA: DE = Hex of numeric string pointed to by HL.
On exit, HL points to first non digit character.
C80F: <CLEAR n,N> where n=string space, N=top of BASIC RAM.
C855: <RUN>
C858: Go address for >Log BASIC programs. The program must have Line Number 0 and FILE Type must be <B0H>
C861: <GOSUB>
C872: <GOTO>
C888: UL ERROR
C890: <RETURN>
C899: RG ERROR
C8B5: <DATA>
C8B7: <REM>
C8CC: <LET>
C91F: Moves a variable (4 bytes) from ACC to (HL). On exit, HL points to the delimiter and DE points to the first of the 4 bytes in the variable area.
C926: <ON>
C944: <IF>
C968: <PRINT>
C9B2: Do a CR-LF unless cursor is in the first column.
C9BF: Do a CR-LF.
C9C9: Delay by sending number of NULL characters in (0141).
C9F1: Here for <TAB()> or <SPC()> initially.
CA01: <TAB()> 
CA04: <SPC()>
CA1B: "?REDO FROM START<CR><LF>" message string.
CA43: <INPUT>
CA72: <READ>
CAFF: "?EXTRA IGNORED<CR><LF>" message string.
CB10: Used by <READ>.
CB34: <NEXT>
CB7F: Calls CB93 to evaluate BASIC expression - check for TM Error.
CB82: Checks for NTF=0 == Numeric Operation.
CB83: Checks for NTF=1 == String Operation.
CB8A: TM ERROR
CB8F: Tests for left bracket and evaluates the expression. Jumps to ?SN ERROR if left bracket not found.
CB93: Evaluates BASIC expression pointed to by HL.
CC0A: Used by several routines (esp.CB93) to test for +,-,",<NOT>,<FN> and evaluates them if found. Also indexes into Jump Table at C0C6 to get addresses for <SGN> thru <MID$>
CC14: MO ERROR
CC45: Evaluate expression then check for right bracket ";", else ?SN ERROR.
CC5E: Loads ACC with variable - sets NTF. HL should point to variable name.
CC6F: All commands <SGN> to <PEEK> and <LEN> to <MID$> pass through here. Numeric routines :<SGN> to <PEEK> jump to CC96. "String" routines :<LEN> to <MID$> continue on to CC7D.
CC96: <USR> and others, jump from here.
CCA7: <OR> ACC = (SP) OR ACC.
CCA8: <AND> ACC = (SP) AND ACC.
CD2F: <NOT>
CD4F: <DIM>
CD54: This routine searches for a variable (name pointed to by HL) and either (1) Returns its address in DE or (2) Creates the variable if not found.
CDF8: Make ACC point to "READY".
CE4D: DD ERROR
CE5E: BS ERROR
CEE9: <FRE>
CF17: <FOS>
CF1A: ACC = Floating point of value in A.
CF1F: <DEF>
CF42: <FN>
CF80: Checks to see if Sorcerer is in RUN mode or DIRECT mode.
CF89: ID ERROR
CF9F: <STR$>
CFC4: Loads (01A2)=String Length in A. (01A4)=Start address of string in DE.
CFD3: Counts number of characters in a string. On entry: HL should point to start of ASCII string. On exit : BC = string length HL = address of delimiter.
D00F: ST ERROR
D015: Prints a string using C585. Again, HL must point to start of string.
D02E: First checks if there's still free string space. ?OS ERROR if none. After a CALL D02E, DE points to start of sub-string. A = sub-string length.
D093: String Space Garbage Collection routine.
D10F:
D146: Sets up registers before continuing to D14F. On entry : Return address - pointer to 4 bytes of string information are on the Stack.
D14F: Used by string commands to remove sub-strings.  
On entry : L=sub-string length.  
BC=start address of sub-string within main 
DE=start address of new string.

D159:  
D17A:  
D18B: <LEN>  
D19A: <ASC>  
D1AB: <CHR$>  
D1BB: <LEFT$>  
D1EB: <RIGHT$>  
D1F5: <MID$>  
D225: <VAL>  
D240: Tests for a right bracket ")") after RIGHT$, LEFT$ or MID$  
On return, B=substring length.  
D24A: <INF>  
D256: <OUT>  
D25C: <WAIT>  
D27A: Used by <OUT> to evaluate - load the port value.  
D28D: A = Hex of numeric string pointed to by HL.  

FC ERROR if > 255.  
D2A1: Writes 2 bytes to tape. (Byte is in A).  
D2A4: Writes 1 byte to tape.  
D2AD: Reads 1 byte from tape.  
D2BD: Writes 1 byte to tape.  
D2C9: <CSAVE>  
D2CE: <CSAVE*>  
D304: Finish off CSAVE.  
D310: Put program FILENAME (max.5 chars) into 16 byte tape  
output file header which starts at (IY+47H).  
D336: If length of FILENAME < 5 then pad to right with blanks.  
D341: <CLOAD>  
D346: <CLOAD*>  
D384: This is part of <CLOAD>. Loads (01B7) with the end of the  
CLOADed program, prints "READY" and re-creates the link  
pounters.  
D394: Repair the link pointers in a BASIC program.  
D39D: ACC = ACC + 0.5  
D3A0: ACC = (HL) + ACC : ADDITION  
D3A6: ACC = (HL) - ACC : SUBTRACTION  
D3AC: ACC = BCDE - ACC  
D3AF: ACC = BCDE + ACC  
D415: Sets ACC = Zero by making the floating point exponent 
(01C2)=0.  
D451: OV ERROR  
D462: BCDE = - BCDE  
D4AB: <LOG>  
D4E3: ACC = ACC * LOG(2)  
D4EA: ACC = (SP) * ACC  
D4EC: ACC = BCDE * ACC  
D54B: ACC = (SP) * ACC  
D54D: ACC = BCDE / ACC  
D5EO: ACC = ACC * 10  
D5F7: Checks if ACC = 0 ? and sets Z flag if it is.  
D606: <SGN>
D60E: ACC = Floating Point of BADE. This routine can be used to return values to the USR(x) function.
B = Exponent
A = MSB including Sign Bit
D = NSB
E = LSB.
For -65536 < x < +65536 : XOR A 
\( \text{LD } B, 98H \) \( A=0 \) for +ve \( \text{LD } DE, NNn \) \( A=FF \) for -ve
JP D60E

D61C: <ABS>
D620: Part of ABS routine, the only difference is that it doesn't test the ACC for zero.
D628: (SP) = ACC. Loads the floating point value from the ACC to the stack To get it back, POP BC then POP DE.
D635: ACC = (HL). Loads floating point value (4 bytes) from HL to ACC.
D638: ACC = BCDE.
D643: BCDE = ACC
D646: BCDE = (HL)
D64F: (HL) = ACC
D652: (HL) = (DE). Move 4 bytes from location pointed to by DE to HL.
D654: (HL) = (DE). Move # bytes in B register.
D65D: ACC = - ACC
D672: Compares ACC with BCDE.
  If ACC < BCDE then A = FFH (-1)
  ACC = BCDE then A = 00
  ACC > BCDE then A = 01
D6CA: <INT>
D6FE: <PEEK>
D705: <POKE>
D716: ACC = Floating Point of Numeric String pointed to by HL.
D7C6: Numeric String = ACC. (Opposite of D716).
D890: Compares ACC with 999,999.
D89F: Floating point 0.5 (00/00/00/00/80).
D8BA: <SQR>
D8C3: ACC = (SP) ^ ACC (POWER)
D908: <EXP>
D948: Table of floating point constants used by EXP - SQR.
D95D: FF/FF/7F/7F = 0.5
  00/00/80/81 = -1
  00/00/00/81 = +1
D999: <RND>
DA02: Tables of data used by RND.
DA0E: <COS> = SIN(x + PI/2)
DA14: <SIN> = x - x^3/3! + x^5/5! - x^7/7! + x^9/9!
  However, the actual expansion used is:
  \( \text{SIN}(2\text{PI}x) = 2\text{PI}x - (2\text{PI}x)^3/3! + (2\text{PI}x)^5/5! - (2\text{PI}x)^7/7! + (2\text{PI}x)^9/9! \)
  Only the first 5 terms of the series are required for 6 digit precision. The above coefficients are located from DA61 to DA74.
DA58: Table of floating point constants (4 bytes each) used by SIN - COS.
DB/0F/49/81 = 1.5708 radians = 90 degrees.
00/00/00/7F = 0.25

DA60: 05 = Number of terms required to calc. SIN - COS
64/26/99/87 = -76.575 = (2PI)^7/7! exactly.
58/34/23/87 = +81.6022 = (2PI)^5/5! approx.
E0/5D/A5/86 = -41.3417 = (2PI)^3/3! exactly.
DA/0F/49/83 = + 6.28319 = (2PI) = 360 degrees.

DA75: <TAN> = <SIN> / <COS>
DABA: <ATN>

DAB1: Table of floating point constants used by ATN.
DAD6: [ EMPTY ROM from here till DFFA !! ]
DFFA: BASIC Warm Start, jumps to C06B.
DFFD: BASIC Cold Start, jumps to C000.
### 8.02 BASIC ROMPAC WORK AREA

<table>
<thead>
<tr>
<th>ADDR</th>
<th>SIZE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>100:</td>
<td>3</td>
<td>Z80 jump instruction to Basic warm start: C3 6B C0 Used if work area and program are saved as a CP/M .COM file.</td>
</tr>
<tr>
<td>103:</td>
<td>3</td>
<td>Z80 jump instruction to USR() function routine. Default address is to the FC ERROR routine. The user can insert the address of his routine with: POKE 260,AD AND 255 : POKE 261,AD / 256</td>
</tr>
<tr>
<td>106:</td>
<td>3</td>
<td>OUT (nn),A instruction. Used by the OUT I,J command.</td>
</tr>
<tr>
<td>109:</td>
<td>14</td>
<td>Fast 4 byte subtract routine used by floating point divide for speed.</td>
</tr>
<tr>
<td>117:</td>
<td>35</td>
<td>Pseudo-random number data used as tables and counters by Basic's RND function.</td>
</tr>
<tr>
<td>13A:</td>
<td>4</td>
<td>Last psuedo-random number generated by RND, in floating point, for RND(0).</td>
</tr>
<tr>
<td>13E:</td>
<td>3</td>
<td>IN A,(nn) instruction. Used by INP() function.</td>
</tr>
<tr>
<td>141:</td>
<td>1</td>
<td>Number of ASCII nulls to print after a carriage return. Defaults to 1.</td>
</tr>
<tr>
<td>142:</td>
<td>1</td>
<td>Terminal line length. Defaults to 64.</td>
</tr>
<tr>
<td>143:</td>
<td>1</td>
<td>Column number of last PRINT with comma field, ie. the start of the last 14 character field.</td>
</tr>
<tr>
<td>144:</td>
<td>1</td>
<td>CTRL 0 output suppression flag. OH = output. OFFH = no output.</td>
</tr>
<tr>
<td>145:</td>
<td>2</td>
<td>Pointer to top of Basic stack.</td>
</tr>
<tr>
<td>147:</td>
<td>2</td>
<td>Current line number. OFFFEH during initialization. OFFFFH in direct mode.</td>
</tr>
<tr>
<td>149:</td>
<td>2</td>
<td>Pointer to start of Basic program text, ie. 01D5H.</td>
</tr>
<tr>
<td>14B:</td>
<td>67</td>
<td>Terminal input buffer. Starts with a comma. Also direct command line.</td>
</tr>
<tr>
<td>18E:</td>
<td>1</td>
<td>Current terminal column position.</td>
</tr>
<tr>
<td>18F:</td>
<td>1</td>
<td>0 when used by CD54 to locate - create variables. &lt;&gt; 0 when used by &lt;DIM&gt;.</td>
</tr>
<tr>
<td>190:</td>
<td>1</td>
<td>Number Type Flag. (See Note 3 above).</td>
</tr>
</tbody>
</table>
191: 1 Multi-Statement Line Flag. 
= 0 if multi-statement line 
<> 0 if only one statement on line.

192: 2 Pointer to highest RAM location, ie. HIMEM.

194: 18 Internal pointers used for string constant, variable 
and string space maintenance.

1A2: 2 Length of string that was just printed.

1A4: 2 Start address of string that was just printed.

1A6: 2 Pointer to top of free string space. Set by CLEAR n command.

1A8: 2 Internal pointer used in string space garbage 
collection.

1AA: 2 Current DATA line number.

1AC: 1 Used by <FOR> and <FN>. 
= 64H when used by <FOR> 
= 80H when used by <FN>

1AD: 1 Used by <RETURN>. Should = 0 for return. Also, 
last character entered from keyboard during input.

1AE: 1 Used by <INPUT> - <READ> to distinguish between them. 
= 0 for <INPUT>. 
<>0 for <READ>.

1AF: 1 Temporary storage for HL. Sometimes used to store 
current or last position reached in a line of BASIC 
before extra processing was required. Also used to 
store pointer to value of variable in variable area.

1B1: 2 Pointer to end of current line being processed by 
BASIC. This can be useful for passing names instead 
of numbers as arguments of the USR() function. 
Points to instruction in the Basic program about to 
be executed when CTRL-C is used to stop execution.

1B3: 2 Line number in program. Set by CTRL-C or STOP.

1B5: 2 Pointer to current statement in program to be 
executed next.

1B7: 2 Pointer to start of variable space at end of program.

1B9: 2 Pointer to start of array space.

1BB: 2 Pointer to end of RAM memory in use, ie. end of 
variable storage space.
1BD: 2   Pointer to current item in DATA list.
1BF: 4   Floating point numeric accumulator.
1C3: 17  Internal storage used for floating point printout and multiplication.
1D4: 1   Zero to signify end of imaginary first program line.
1D5: x   Start of Basic program storage.
CHAPTER 9 --- MONITOR

9.01 MONITOR POKE ADDRESSES

>SE S=xx SCREEN SPEED 8K 16K 32K 48K
  S=0 FASTEST. POWER-ON DEFAULT.
  S=10 SLOWER.

>SE T=xx BAUD RATE 8K 16K 32K 48K
  T=0 1200 BAUD FOR SERIAL PORT AND CASSETTE TAPE.
  EXAMPLE: POKE 32718,64 :REM 1200 BAUD.
  T=1 300 BAUD FOR SERIAL PORT AND CASSETTE TAPE.
  EXAMPLE: POKE 32718,0 :REM 300 BAUD.

>SE O=yy OUTPUT PORT 8K 16K 32K 48K
  V - VIDEO DRIVER 0E01BH POKE 32720,27 :POKE 32721,224
  P - PARALLEL OUT 0E021H POKE 32720,33 :POKE 32721,224
  L - CENTRONICS 0E993H POKE 32720,147:POKE 32721,233
  S - CASSETTE OUT 0E012H POKE 32720,18 :POKE 32721,224
  xxyy - USER DEFINED xxyy POKE 32720,yy :POKE 32721,xx

>SE I=xx INPUT PORT 8K 16K 32K 48K
  K - KEYBOARD 0E018H POKE 32722,24 :POKE 32723,224
  P - PARALLEL IN 0E01EH POKE 32722,30 :POKE 32723,224
  S - CASSETTE IN 0E00FH POKE 32722,15 :POKE 32723,224
  xxyy - USER DEFINED xxyy POKE 32722,yy :POKE 32723,xx
**9.02 MONITOR WORKAREA**

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+00</td>
<td>91</td>
<td>60 BYTE MONITOR COMMAND BUFFER. ASCII TEXT IS TERMINATED BY A CARRIAGE RETURN, ODH.</td>
</tr>
<tr>
<td>+3C</td>
<td>CD</td>
<td>PORT 0FEH STATUS.</td>
</tr>
<tr>
<td>+3D</td>
<td>CE</td>
<td>BAUD RATE. 1200 BAUD= 040H. 300 BAUD= 00 HEX.</td>
</tr>
<tr>
<td>+3E</td>
<td>CF</td>
<td>SEND DELAY TIME FROM &gt;SE S=nn COMMAND.</td>
</tr>
<tr>
<td>+3F</td>
<td>D0</td>
<td>CURRENT SEND ROUTINE ADDRESS FROM &gt;SE 0=xxyy.</td>
</tr>
<tr>
<td>+41</td>
<td>D2</td>
<td>CURRENT RECEIVE ROUTINE ADDRESS FROM &gt;SE I=xxyy.</td>
</tr>
<tr>
<td>+43</td>
<td>D4</td>
<td>BATCH MODE STATUS. 0=NORMAL 1=BATCH MODE.</td>
</tr>
<tr>
<td>+44</td>
<td>D5</td>
<td>MONITOR OUTPUT PROMPT OF '&gt;'.</td>
</tr>
<tr>
<td>+45</td>
<td>D6</td>
<td>BAUD RATE - MOTOR CONTROL 10H=MOTOR1 20H=MOTOR2</td>
</tr>
<tr>
<td>+46</td>
<td>D7</td>
<td>TAPE INPUT AND OUTPUT CRC CHECK DATA BYTE.</td>
</tr>
<tr>
<td>+47</td>
<td>D8</td>
<td>FIVE LETTER OUTPUT FILE NAME FROM &gt;SA AND CSAVE</td>
</tr>
<tr>
<td>+4C</td>
<td>DD</td>
<td>OUTPUT FILE HEADER ID. USUALLY 55H.</td>
</tr>
<tr>
<td>+4D</td>
<td>DE</td>
<td>OUTPUT FILE TYPE FROM &gt;SE F=nn COMMAND.</td>
</tr>
<tr>
<td>+4E</td>
<td>DF</td>
<td>2-BYTE LENGTH OF FILE IN BYTES.</td>
</tr>
<tr>
<td>+50</td>
<td>E1</td>
<td>2-BYTE PROGRAM LOAD ADDRESS. BASIC=01D5H.</td>
</tr>
<tr>
<td>+52</td>
<td>E3</td>
<td>2-BYTE PROGRAM &gt;GO ADDRESS. &gt;SE X=xxyy COMMAND.</td>
</tr>
<tr>
<td>+54</td>
<td>E5</td>
<td>3 RESERVED BYTES.</td>
</tr>
<tr>
<td>+57</td>
<td>E8</td>
<td>FIVE LETTER INPUT FILE NAME FROM &gt;LO AND CLOAD.</td>
</tr>
<tr>
<td>+5C</td>
<td>ED</td>
<td>INPUT FILE HEADER ID.</td>
</tr>
<tr>
<td>+5D</td>
<td>EE</td>
<td>INPUT FILE TYPE.</td>
</tr>
<tr>
<td>+5E</td>
<td>EF</td>
<td>2-BYTE LENGTH OF INPUT FILE IN BYTES.</td>
</tr>
<tr>
<td>+60</td>
<td>F1</td>
<td>2-BYTE PROGRAM LOAD ADDRESS FROM TAPE.</td>
</tr>
<tr>
<td>+62</td>
<td>F3</td>
<td>2-BYTE PROGRAM &gt;GO ADDRESS FROM TAPE.</td>
</tr>
<tr>
<td>+64</td>
<td>F5</td>
<td>3 RESERVED BYTES.</td>
</tr>
<tr>
<td>+67</td>
<td>F8</td>
<td>CHARACTER UNDER THE CURSOR.</td>
</tr>
<tr>
<td>+68</td>
<td>F9</td>
<td>2-BYTE SCREEN OFFSET FOR ROW CURSOR IS ON.</td>
</tr>
<tr>
<td>+6A</td>
<td>FB</td>
<td>2-BYTE COLUMN NUMBER CURSOR IS IN [0-63].</td>
</tr>
<tr>
<td>+6C</td>
<td>FD</td>
<td>LAST CHARACTER FROM KEYBOARD FOR REPEAT FUNCTION.</td>
</tr>
<tr>
<td>+6D</td>
<td>FE</td>
<td>2 BYTES OF RESERVED SPACE.</td>
</tr>
</tbody>
</table>
### 9.03 Useful Monitor Routines and Their Functions

<table>
<thead>
<tr>
<th>ADDR</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E000</td>
<td>Monitor Cold Start.</td>
</tr>
<tr>
<td>E003</td>
<td>Monitor Warm Start. Jump to here from your programs.</td>
</tr>
<tr>
<td>E006</td>
<td>Relocate Monitor Stack to Address Passed in Register HL.</td>
</tr>
<tr>
<td>E009</td>
<td>Receive: Return character in 'A' and 'NZ' flag.</td>
</tr>
<tr>
<td>E00C</td>
<td>Send: Send character in 'A' to current output device.</td>
</tr>
<tr>
<td>E00F</td>
<td>Serial In: Read character from UART.</td>
</tr>
<tr>
<td>E012</td>
<td>Serial Out: Send character to UART.</td>
</tr>
<tr>
<td>E015</td>
<td>Quick Check: Return 'NZ' if 'CTRL-C', 'RUN/STOP' or 'ESC'</td>
</tr>
<tr>
<td>E018</td>
<td>Keyboard: Routine for receive if ( &gt;SE I=K ). Keyboard Scan</td>
</tr>
<tr>
<td>E01B</td>
<td>Video: Routine for send if ( &gt;SE O=V ). Video Display</td>
</tr>
<tr>
<td>E021</td>
<td>Parallel Out: Send character to parallel port.</td>
</tr>
<tr>
<td>E024</td>
<td>Centronics Out: Handshake character to Centronics Printer</td>
</tr>
<tr>
<td>E027</td>
<td>Turn Cassette Motor On. Register B contains 1 or 2.</td>
</tr>
<tr>
<td>E02A</td>
<td>Tape Save Routine.</td>
</tr>
<tr>
<td>E02D</td>
<td>Tape Load Routine.</td>
</tr>
<tr>
<td>E13A</td>
<td>Monitor Input Routine Fills 60 Byte Input Buffer.</td>
</tr>
<tr>
<td>E1A2</td>
<td>Find Monitor Workarea and Put Base Address in Register IY</td>
</tr>
<tr>
<td>E1BA</td>
<td>Message: Output Text String That Ends With a Zero Byte.</td>
</tr>
<tr>
<td>E1C9</td>
<td>Error: Print Error Message and Diagnostic Message.</td>
</tr>
<tr>
<td>E1D4</td>
<td>Over: Process the Batch Mode Over Command.</td>
</tr>
<tr>
<td>E1E8</td>
<td>Print Hexadecimal Number in Register 'DE' in Hex.</td>
</tr>
<tr>
<td>E1ED</td>
<td>Print Hexadecimal Number in Register 'A' in Hex.</td>
</tr>
<tr>
<td>E205</td>
<td>Print Carriage Return and Line Feed.</td>
</tr>
<tr>
<td>E22F</td>
<td>Parse Input Command String.</td>
</tr>
<tr>
<td>E23D</td>
<td>Put ASCII Hex Number in Register 'DE'. Reverse of E1E8H.</td>
</tr>
<tr>
<td>E2D2</td>
<td>Send the Number of Blanks in Register 'B'.</td>
</tr>
<tr>
<td>E423</td>
<td>Dump Command Processor.</td>
</tr>
<tr>
<td>E538</td>
<td>Enter Command Processor.</td>
</tr>
<tr>
<td>E562</td>
<td>Move Command Processor.</td>
</tr>
<tr>
<td>E597</td>
<td>Go Command Processor.</td>
</tr>
<tr>
<td>E5A2</td>
<td>Set Command Processor.</td>
</tr>
<tr>
<td>E638</td>
<td>Save Command Processor.</td>
</tr>
<tr>
<td>E6B9</td>
<td>Files Command Processor.</td>
</tr>
<tr>
<td>E78A</td>
<td>Load Command Processor.</td>
</tr>
<tr>
<td>E858</td>
<td>Batch Command Processor.</td>
</tr>
<tr>
<td>E85C</td>
<td>Create Command Processor.</td>
</tr>
<tr>
<td>E884</td>
<td>List Command Processor.</td>
</tr>
<tr>
<td>E8A1</td>
<td>Test Command Processor.</td>
</tr>
<tr>
<td>E98A</td>
<td>PP Command Processor.</td>
</tr>
<tr>
<td>E9B1</td>
<td>Clear the Display and Rewrite the Graphic Character Set.</td>
</tr>
<tr>
<td>E9CC</td>
<td>Move Cursor To Row and Column Number in MWA+68 and MWA+6A</td>
</tr>
<tr>
<td>E9D6</td>
<td>Find the Cursor Row and Column Numbers.</td>
</tr>
<tr>
<td>EB10</td>
<td>Replace Character Under the Cursor.</td>
</tr>
<tr>
<td>EC1E</td>
<td>Keyboard Input Tables to Decode the ASCII Value to Return</td>
</tr>
<tr>
<td>EDFE</td>
<td>Character Set for the First 64 Graphic Characters.</td>
</tr>
</tbody>
</table>
9.04 RELOCATE THE MONITOR STACK.

>EN 0
0000: 21 yy xx C3 06 E0 /
>GO 0

The monitor stack on power up builds from the last memory location downward. In a 32K machine this is at 7FFFH. The stack can be relocated elsewhere in memory by providing a different last address in yy xx of the above code.

Example: To relocate to 6FFFH, use

21 FF 6F C3 06 E0 /

=================================================================================

9.05 ERASE MEMORY BY FILLING IT WITH ZEROES.

BYE EXIT TO THE MONITOR.
>EN 0 BYTE 0 CONTAINS THE FILL CHARACTER.
0000: 0 /
>MO 0 7F00 1 ZERO MEMORY THROUGH ADDRESS 7F00 HEX.

=================================================================================

9.06 EXECUTE MONITOR COMMANDS FROM A BASIC PROGRAM

100 C$ = "LO 1 3800"+CHR$(13) : REM TYPICAL COMMAND
110 M = PEEK(-4096)+PEEK(-4095)*256-111
120 IF M>32767 THEN M=M-65536
130 FOR I=1 TO LEN(CM$)
140 POKE M+I,ASC(MID$(C$,I,1)) : NEXT I :REM PUT IN BUFFER
150 POKE 260,138:POKE 261,231:X=USR(0) :REM DO COMMAND
CHAPTER 10 --- M. CODE INTERFACE

10.01 PROTECT MEMORY FOR MACHINE LANGUAGE ROUTINES

1. 0000 - 00FF HEX IS NEVER USED BY THE ROMPAC BASIC.
2. RELOCATE THE STACK. USE THE MEMORY ABOVE THE STACK.
3. CLEAR xx,yy CREATES A WINDOW ABOVE THE STRING SPACE.

10.02 CALL A MACHINE LANGUAGE ROUTINE

100 POKE 260, ADDR AND 255 : POKE 261, ADDR/256
110 X = USR(0)

ADDR CONTAINS THE ADDRESS OF THE ROUTINE.
PLACE THE ADDRESS OF THE ROUTINE IN BYTES 260 AND 261.
INVOKE THE CALL TO THE ROUTINE WITH THE USR() FUNCTION.
A 'RET' INSTRUCTION IN THE ROUTINE WILL RETURN TO BASIC.

10.03 FASTER USR() PARAMETER PASSING.

100 J = USR(0) : J = PEEK(X)
110 J = PEEK(USR(X))

LINE 110 IS EQUIVALENT TO LINE 100, BUT FASTER.

10.04 POKE MACHINE HEX CODE INTO MEMORY.

100 DEF FNA(X) = (X AND 15) - 9 * (X > 64)
110 READ J : REM FIRST DATA ITEM IS # OF BYTE TO POKE
120 :
130 FOR I = 0 TO J - 1 : READ A$
140 K = FNA(ASC(A$)) * 16 + FNA(ASC(RIGHT$(A$,1)))
150 POKE OFFSET + I, K : NEXT I
160 :
200 DATA 6,11,00,00,C3,EB,E1 : REM EXAMPLE HEX CODE

THIS ROUTINE CONVERTS HEX CODE INTO DECIMAL TO POKE.
The CODE LOADS BEGINNING AT THE ADDRESS OF 'OFFSET'.

10.05 PASS ARGUMENT IN 'A' REGISTER TO MACHINE CODE

100 POKE 262,195 : POKE 264, ADDR/256
110 OUT ADDR AND 255, X

ADDR CONTAINS THE ADDRESS OF THE ROUTINE.
The 'OUT' COMMAND INVOKES THE CALL TO THE ROUTINE.
ARGUMENT X IS PASSED TO THE ROUTINE IN THE ACCUMULATOR.
10.06 PASS TWO PARAMETERS USING THE OUT I,J INSTRUCTION.

CHANGE THE 'OUT I,J' INSTRUCTION TO A RST OH COMMAND. THE RST OH IS A MACHINE LANGUAGE RESTART TO ADDRESS 0. IT IS JUST LIKE A CALL INSTRUCTION, EXCEPT IT OCCUPIES ONLY ONE BYTE OF MEMORY.

100 POKE 262,199 : REM 'OUT' IS NOT A 'RST OH'
110 OUT I,J : REM CALL YOUR ROUTINE AT ADDRESS 0.

YOUR MACHINE LANGUAGE ROUTINE MUST RESIDE AT ADDRESS 0. THE I PARAMETER FROM THE OUT COMMAND IS PASSED IN ADDRESS 0107H, AND THE J PARAMETER IS PASSED IN THE Z80 ACCUMULATOR. TO RETURN TO BASIC, YOU MUST POP THE RST RETURN ADDRESS OFF OF THE STACK SO THAT THE RETURN ADDRESS FOR THE 'OUT' COMMAND IS USED INSTEAD.

===================================

10.07 PASS ROUTINE ADDRESS IN USR() FUNCTION CALL.

0000: CD D0 C7 ENTRY CALL 0C7D0H ;GET USR() #
0003: D5 PUSH DE ;CALL ADDRESS
0004: C9 RET ;GOTO ROUTINE

100 POKE 260,0 : POKE 261,0 : REM USR() ENTRY ADDRESS
110 X = USR(AD) : REM JUMP TO MEMORY ADDRESS IN AD

THE USR() FUNCTION WILL ENTER THE ROUTINE AT ADDRESS 0, AND THEN JUMP THE ADDRESS PASSED AS A PARAMETER IN THE VARIABLE AD. YOUR CODE AT ADDRESS AD WOULD END WITH A NORMAL 'RET' TO RETURN TO BASIC.

IF YOUR ROUTINE USES REGISTER HL, YOU SHOULD PUSH HL UPON ENTRY, AND THEN POP HL JUST BEFORE RETURNING.

===================================

10.08 MULTIPLE USR() ROUTINES SELECTED BY LETTER IN ()

100 POKE 260,0 : POKE 261,0 : REM ONLY ONE ENTRY ADDRESS.
110 :
120 X = USR(A) : REM EXECUTE ROUTINE "A"
130 X = USR(B) : REM EXECUTE ROUTINE "B"

THE ADDRESSES FOR THE ABOVE ROUTINES ARE FOUND IN A TABLE. THE TABLE CONTAINS THE MATCH LETTER, SUCH AS "A", FOLLOWED BY THE ADDRESS OF THE ROUTINE. A ZERO BYTE IN THE MATCH LETTER POSITION WILL TERMINATE THE TABLE.
0000: CD A2 E1 ENTRY CALL 0E1A2H ;GET IY
0003: 2A B1 01 LD HL,(01B1H) ;BASIC POINTER
0006: 3E B2 DEC HL ;POINTER BACK
0008: 2B LP1 DEC HL ;FOUNDED USR( YET
0009: BE CF (HL) ;POINTER BACK
000A: 20 FC JR NZ,LP1-$ ;KEEP LOOKING
000C: 23 INC HL ;SKIP OVER
000D: 23 INC HL ;POINT LETTER
000E: 7E LD A,(HL) ;GET LETTER
000F: 21 27 00 LD HL, TABLE ;TABLE BASE
0012: BE CF (HL) ;MATCH ??
0013: 28 0C JR Z,FOUND-$ ;JUMP IF MATCH
0015: 47 LD B,A ;SAVE LETTER
0016: AF XOR A ;LOAD ZERO
0017: BE CF (HL) ;TABLE END ??
0018: CA E5 C7 JP Z,OC7E5H ;FC ERROR
001B: 78 LD A,B ;RESTORE LETTER
001C: 23 INC HL ;SKIP LETTER
001D: 23 INC HL ;SKIP yy ADDR
001E: 23 INC HL ;SKIP xx ADDR
001F: 18 F1 JR LP2-$ ;TRY NEXT MATCH
0021: 23 FOUND INC HL ;SKIP LETTER
0022: 7E LD A,(HL) ;GET yy ADDRESS
0023: 23 INC HL ;SKIP yy ADDR
0024: 66 LD H,(HL) ;GET xx ADDRESS
0025: 6F LD L,A ;HL = xxyy ADDR
0026: E9 JP (HL) ;GOTO ROUTINE
0027: 41 TABLE DEFB 'A' ;MATCH LETTER
0028: yy xx DEFW A-LABEL ;ROUTINE ADDR
0029: 42 DEFB 'B' ;MATCH LETTER
002A: yy xx DEFW B-LABEL ;ROUTINE ADDR
002B: 00 DEFB 00 ;TABLE END FLAG

================================================================================================

10.09 UP-LOADER FOR MACHINE LANGUAGE ROUTINES

THIS SHORT ROUTINE IS USEFUL FOR MOVING MACHINE CODE TO ITS PROPER ORG ADDRESS AFTER BEING LOADED AT A DIFFERENT ADDRESS.

0100: 01 yy xx ENTER LD BC,LENGTH ;# OF BYTES TO MOVE
0103: 11 yy xx LD DE,TO ;DESTINATION ADDRESS
0106: 21 yy xx LD HL,PROM ;SOURCE ADDRESS
0109: ED B0 LDIR ;BLOCK MOVE
010B: C3 yy xx JP START ;PROGRAM START

THIS ROUTINE IS PARTICULARLY USEFUL IN CP/M ENVIRONMENTS WHERE THE CODE HAS BEEN MOVED DOWN TO 0110H WITH THE MONITOR AND THEN SAVED ON DISK. WHEN THE PROGRAM IS LOADED FROM DISK, CP/M EXECUTES THIS UP-LOADER WHICH LOADED JUST AHEAD OF THE PROGRAM. THE PROGRAM IS MOVED TO ITS PROPER ADDRESS AND EXECUTED.
CHAPTER 11 --- M. CODE ROUTINES

11.01 ROW - COLUMN ROTATION OF A CHARACTER CELL

ENTER WITH AN ASCII CHARACTER IN REGISTER A.

THE BIT PATTERN FOR THE CHARACTER IS CONVERTED FROM ITS ROW BY ROW FORMAT INTO COLUMN BY COLUMN DOT DATA THAT CAN BE USED TO FIRE EIGHT HAMMERS OF A GRAPHICS PRINTER.

TO ACTIVATE, SET THE OUTPUT VECTOR TO THIS DRIVER. \(>SE 0=6500\)

```
ORG 6500H
PUSH BC
PUSH DE
PUSH HL
LD LA
LD H,0 ;HL = ASCII CHAR
ADD HL,HL ;*2
ADD HL,HL ;*4
ADD HL,HL ;*8
LD DE,0F800H ;ASCII TABLE BASE
ADD HL,DE ;TABLE POINTER
LD E,8 ;COL LOOP COUNTER
LD C,80H ;AND MASK BIT 7
LD D,0 ;CLEAR D
PUSH HL ;SAVE HL NEXT LOOP
LD B,8 ;ROW LOOP COUNTER
LD A,(HL) ;GET PATTERN
AND C ;MASK BIT OFF
RLCA ;COMBINE
LD D,A ;ROTATE A LEFT
INC HL ;RESAVE IN D
DJNZ LOOP2-$ ;NEXT ROW PATTERN
DJNZ LOOP3-$ ;GET 8 ROW BITS
LD B,E
RRCA
DJNZ LOOP3-$ ;ROTATE BACK
OUT (255),A ;SEND COL TO PRINTER
RRC C ;ROTATE MASK
POP HL ;ADDRESS BASE
DEC E
JR NZ,LOOP1-$ ;DO 8 COLUMNS
POP NZ
POP BC
RET ;BACK TO MAIN PROG
```
11.02 IMAGE DRIVER TO CONVERT TO \texttt{>SA} FILES.

This is a simple, yet very useful, driver that converts any video output to a memory image. Two frequent uses of this driver are: convert a Basic program into its ASCII file, and convert a disassembler listing into a word processor file.

\begin{verbatim}
0000: E5 ENTRY PUSH HL
0001: 2A 0D 00 LD HL,(POINT) ;GET POINTER
0004: 77 LD (HL),A ;CREATE IMAGE
0005: 23 INC HL ;NEXT MEM CELL
0006: 22 0D 00 LD (POINT),HL ;PUT POINTER
0009: E1 POP HL
000A: C3 1B EO JP OE01BH ;TO VIDEO TOO
000D: yy xx POINT DEFW xxyy ;START ADDRESS
\end{verbatim}

EXAMPLE: CREATE A WORD PROCESSOR FILE FROM DISASSEMBLER.

1. INSERT WORD PROCESSOR ROMPAC.

2. USE THE MONITOR TO LOAD A MACHINE LANGUAGE DISASSEMBLER SUCH AS:\texttt{>LO DIS32}

3. LOAD THE ABOVE DRIVER.

4. START ADDRESS IS \texttt{80FH}:

5. CHANGE OUTPUT VECTOR:\texttt{>SE 0=0000}

6. EXECUTE DISASSEMBLER.\texttt{>GO 6A00}

7. DISASSEMBLE ROUTINE OF INTEREST.

8. EXIT DISASSEMBLER BACK TO MONITOR.

9. RESTORE VIDEO VECTOR.\texttt{>SE 0=V}

10. FIND FINAL ADDRESS IN (POINT).\texttt{>DU D E}

ONE MIGHT SAVE THE FILE AT THIS POINT WITH

\texttt{>SA NAME 080F xxyy}

11. ENTER 03 INTO THIS FINAL ADDRESS. THIS PROVIDES THE END-OF-FILE MARKER FOR THE WORD PROCESSOR.

12. EXECUTE WORD PROCESSOR. \texttt{>PP}

YOUR FILE SHOULD BE PRESENT. IT MAY BE DOUBLE SPACED SINCE THE WORD PROCESSOR CONVERTS BOTH \texttt{<CR>} AND \texttt{<LF>} INTO \texttt{<CR>}s.

13. USE THIS MACRO TO REMOVE DOUBLE SPACING.

\texttt{D1}
\texttt{F1}

\texttt{COMMAND\textgreater A} \texttt{(SAVE MACRO)}
\texttt{COMMAND\textgreater B} \texttt{(START ON LINE TO DELETE)}
\texttt{COMMAND\textgreater A200} \texttt{(EXECUTE MACRO)}
11.03 PINE WOOD DERBY CONTROLLER

The following machine language and basic programs are included as illustrations of how to use the Sorcerer to monitor external events. In this case, photo detectors were mounted in each of three racing lanes for a Cub Scout Pine Wood Derby. As the hand made cars crossed the finish line, the photo detector light source would be interrupted. The Sorcerer would observe the changed state of the photo detectors and stop that car's timer. After all three cars had crossed the finish line, the three timers would be displayed in large block lettering for the anxious audience to read. The resolution of the timing was around 1 milisecond which kept parents and their sons from arguing over whose car was the fastest.

PINE WOOD DERBY CONTROLLER

ORG 4000H
ENTRY JR SKIP-$
CAR1 DEFB 0
CAR2 DEFB 0
CAR3 DEFB 0
COUNT DEFW 0
BUFF EQU 0

DB FFSKIP IN A,(0FFH) ;WAIT FOR START
CB 47BIT 0,A ;GATE TO OPEN
20 FAJR NZ,L4007-$

THE GATE IS OPEN. THE CARS ARE ON THEIR WAY DOWN THE TRACK

LD A,001H
LD (CAR1),A
LD (CAR2),A
LD (CAR3),A ;START ALL TIMERS

LD A,001H
LD (CAR1),A
LD (CAR2),A
LD (CAR3),A

DB FF FINISH IN A,(0FFH) ;WATCH FINISH LINE
CB 4FBIT 1,A ;IS CAR1 ACROSS?
20 07JR NZ,L4025-$ ;JUMP IF NOT
F5PUSH AF
3E 00LD A,000H
32 04 40LD (CAR3),A ;STOP TIMER #1
F1PUS AF

CB 57L4025BIT 2,A ;IS CAR2 ACROSS?
20 07JR NZ,L4030-$ ;JUMP IF NOT
F5PUSH AF
3E 00LD A,000H
32 03 40LD (CAR2),A ;STOP TIMER #2
F1PUS AF
; UPDATE TIMER CLOCK, WHICH IS A SIMPLE INCREMENTING COUNTER.

; GET TIMER.
LD HL,(COUNT) ;HL = 4 DECIMAL

; DIGITS RATHER THAN A BINARY #.
INC A

; HL = HL + 1
ADC A,H

; REPLACE TIMER
LD (COUNT),HL

; UPDATE EACH CAR'S DISPLAY ON CRT WITH NEW COUNT VALUE UNTIL
; CAR'S FINISH FLAG = 0. THE RACE IS OVER WHEN ALL FLAGS = 0.

; IS CAR1 ACROSS?
LD A,(CAR1)

; CAR'S BUFFER
LD DE,BUFF+4H
CALL MOVE

; CAR'S BUFFER
LD DE,BUFF+0CH
CALL MOVE

; CAR'S BUFFER
LD DE,BUFF+14H
CALL MOVE

; NEW CRT DISPLAY
CALL MESSAGE

; NOW CHECK TO SEE IF ALL CARS ARE ACROSS

; CHECK FIRST FLAG
LD HL,CAR1

; CAR2 FINISH FLAG
LD A,(HL)
INC HL
OR A
JR Z,L4062-$

; CAR3 FINISH FLAG
LD A,(HL)
INC HL
OR A
JR Z,L406E-$

; DONE IF ALL = 0
JP NZ,FINISH
RET
This routine loads each car's buffer with the ASCII characters that make up the four decimal digit count in HL.

Enter with a car's unique buffer address in DE.

<table>
<thead>
<tr>
<th>Address</th>
<th>Hex Code</th>
<th>Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>407D:</td>
<td>2A 05 40</td>
<td>MOVE HL, (COUNT) ; GET TIMER VALUE</td>
</tr>
<tr>
<td>4080:</td>
<td>EB</td>
<td>EX DE, HL ; HL = BUFFER ADDR</td>
</tr>
<tr>
<td>4081:</td>
<td>7A</td>
<td>LD A, D ; DE = 4 DIGITS</td>
</tr>
<tr>
<td>4082:</td>
<td>CB 3F</td>
<td>SRL A</td>
</tr>
<tr>
<td>4084:</td>
<td>CB 3F</td>
<td>SRL A</td>
</tr>
<tr>
<td>4086:</td>
<td>CB 3F</td>
<td>SRL A</td>
</tr>
<tr>
<td>4088:</td>
<td>CB 3F</td>
<td>SRL A</td>
</tr>
<tr>
<td>408A:</td>
<td>F6 30</td>
<td>OR 030H ; FIRST ASCII DIGIT</td>
</tr>
<tr>
<td>408C:</td>
<td>77</td>
<td>LD (HL), A ; PUT IT IN BUFFER</td>
</tr>
<tr>
<td>408D:</td>
<td>23</td>
<td>INC HL</td>
</tr>
<tr>
<td>408E:</td>
<td>7A</td>
<td>LD A, D</td>
</tr>
<tr>
<td>408F:</td>
<td>E6 0F</td>
<td>AND 00FH</td>
</tr>
<tr>
<td>4091:</td>
<td>F6 30</td>
<td>OR 030H ; 2ND ASCII DIGIT</td>
</tr>
<tr>
<td>4093:</td>
<td>77</td>
<td>LD (HL), A ; PUT IT IN BUFFER</td>
</tr>
<tr>
<td>4094:</td>
<td>23</td>
<td>INC HL</td>
</tr>
<tr>
<td>4095:</td>
<td>7B</td>
<td>LD A, E</td>
</tr>
<tr>
<td>4096:</td>
<td>CB 3F</td>
<td>SRL A</td>
</tr>
<tr>
<td>4098:</td>
<td>CB 3F</td>
<td>SRL A</td>
</tr>
<tr>
<td>409A:</td>
<td>CB 3F</td>
<td>SRL A</td>
</tr>
<tr>
<td>409C:</td>
<td>CB 3F</td>
<td>SRL A</td>
</tr>
<tr>
<td>409E:</td>
<td>F6 30</td>
<td>OR 030H ; 3RD ASCII DIGIT</td>
</tr>
<tr>
<td>40A0:</td>
<td>77</td>
<td>LD (HL), A ; PUT IT IN BUFFER</td>
</tr>
<tr>
<td>40A1:</td>
<td>23</td>
<td>INC HL</td>
</tr>
<tr>
<td>40A2:</td>
<td>7B</td>
<td>LD A, E</td>
</tr>
<tr>
<td>40A3:</td>
<td>E6 0F</td>
<td>AND 00FH</td>
</tr>
<tr>
<td>40A5:</td>
<td>F6 30</td>
<td>OR 030H ; 4TH ASCII DIGIT</td>
</tr>
<tr>
<td>40A7:</td>
<td>77</td>
<td>LD (HL), A ; PUT IT IN BUFFER</td>
</tr>
</tbody>
</table>

This routine displays a 24 character buffer in the CRT in three rows of eight characters each. The characters are displayed in large block size where each dot occupies a full character cell. The start address of the buffer is BUFF.

<table>
<thead>
<tr>
<th>Address</th>
<th>Hex Code</th>
<th>Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>40A9:</td>
<td>FD E5</td>
<td>MESSAGE PUSH IY</td>
</tr>
<tr>
<td>40AB:</td>
<td>FD 21 00 F1</td>
<td>LD IY, 0F100H ; TOP OF SCREEN</td>
</tr>
<tr>
<td>40AF:</td>
<td>21 00 00</td>
<td>LD HL, BUFF</td>
</tr>
<tr>
<td>40B2:</td>
<td>0E 03</td>
<td>LD C, 003H ; C = # OF ROWS</td>
</tr>
<tr>
<td>40B4:</td>
<td>06 08</td>
<td>NEXTROW LD B, 008H ; B = # OF CHAR</td>
</tr>
<tr>
<td>40B6:</td>
<td>7E</td>
<td>NEXTCOL LD A, (HL) ; GET ASCII CHAR</td>
</tr>
<tr>
<td>40B7:</td>
<td>C5</td>
<td>PUSH BC</td>
</tr>
<tr>
<td>40B8:</td>
<td>E5</td>
<td>PUSH HL</td>
</tr>
<tr>
<td>40B9:</td>
<td>FD E5</td>
<td>PUSH IY</td>
</tr>
<tr>
<td>40BB:</td>
<td>CD D7 40</td>
<td>CALL DISPLAY ; SHOW IN BIG SIZE</td>
</tr>
<tr>
<td>40BE:</td>
<td>FD E1</td>
<td>POP IY</td>
</tr>
<tr>
<td>40C0:</td>
<td>E1</td>
<td>POP HL</td>
</tr>
<tr>
<td>40C1:</td>
<td>C1</td>
<td>POP BC</td>
</tr>
<tr>
<td>40C2:</td>
<td>23</td>
<td>INC HL</td>
</tr>
<tr>
<td>40C3:</td>
<td>11 08 00</td>
<td>LD DE, 8 ; MOVE SCRNL POINTER</td>
</tr>
</tbody>
</table>
; THIS ROUTINE CREATES A LARGE SIZE CHARACTER FROM THE BIT PATTERN FOR THE ASCII CHARACTER. THE CHARACTER IS LOCATED BY HAVING IY POINT TO THE TOP LEFT CORNER OF THE CHARACTER.
; ENTER WITH ASCII CHARACTER IN A, AND SCREEN LOCATION IN IY.

40C6: FD 19 ADD IY,DE
40C8: 10 EC DJNZ NEXTCOL-$ ;NEXT CHARACTER
40CA: 0D DEC C ;THIS ROW DONE
40CB: 28 07 JR Z,L40D4-$ ;ANY MORE TO DO?
40CD: 11 40 02 LD DE,0240H ;MOVE SCRN POINTER
40DF: FD 19 ADD IY,DE ; TO NEXT ROW.
40D2: 18 E0 JR NEXTROW-$
40D4: FD E1 POP IY ;SCREEN COMPLETE
40D6: C9 RET

40D7: OE 08 DISPLAY LD C,008H ;ROW COUNTER
40D9: 6F LD L,A
40DA: 26 00 LD H,000H
40DC: 29 ADD HL,HL
40DD: 29 ADD HL,HL
40DE: 29 ADD HL,HL ;OFFSET = ASCII*8
40DF: 11 00 F8 LD DE,OF800H ;START OF BIT MAP
40E0: 19 ADD HL,DE ;HL = PATTERN ADDR
40E2: 7F CHAR LD A,(HL) ;A = FIRST ROW
40E4: 06 08 LD B,008H ;CELL WIDTH
40E6: 07 COL RLCA ;DETERMINE WHITE
40E7: 38 06 JR C,WHITE-$ ; OR BLACK DOT.
40E9: PD 36 00 20 LD (IY+000H),020H ;BLACK
40ED: 18 04 JR NEXT-$
40EF: PD 36 00 CO WHITE LD (IY+000H),0COH ;WHITE
40F3: PD 23 NEXT INC IY
40F5: 10 EF DJNZ COL-$ ;LOOP CELL WIDTH
40F7: 0D DEC C ;COUNT THIS ROW
40F9: C8 RET Z ;EXIT AFTER 8TH
40FA: 11 38 00 LD DE,038H ;POINT START NEXT
40FC: PD 19 ADD IY,DE ; ROW ON SCREEN.
40FE: 23 INC HL ;NEXT ROW PATTERN
40FF: 18 E2 JR CHAR-$
11.04 PINE WOOD DERBY'S BASIC PROGRAM

100 REM PINE WOOD DERBY
110 :
120 REM CREATE SOLID WHITE GRAPHIC CHARACTER IN ASCII #192
130 :
140 FOR I=1 TO 8:POKE -513+I,255:NEXT I
150 :
160 REM ESTABLISH USR() ADDRESS FOR 4000H.
170 :
180 POKE 261,64
190 :
200 REM CLEAR SCREEN, PUT MESSAGE IN 24 BYTE BUFFER
210 :
220 PRINT CHR$(12):A$="READY GET SET GO . . ."
230 FOR I=1 TO 24:POKE I-1,ASC(MID$(A$,I,1)):NEXT I
240 :
250 REM NOW SEND BUFFER TO CRT IN LARGE LETTERS
260 :
270 POKE 260,169:X=USR(X)
280 :
290 REM SCREEN LOOKS LIKE THIS WITH 3 ROWS OF 8 CHARACTERS:
300 REM
310 REM READY
320 REM GET SET
330 REM GO . . .
340 REM
350 :
360 REM NOW WAIT FOR THE GATE TO OPEN AND START TIMERS
370 REM CALL MACHINE CODE ROUTINE AT 4000H.
380 :
390 POKE 260,0:X=USR(X)
400 :
410 REM RACE IS NOW OVER. SORT TIMES FOR 1ST, 2ND, 3RD PLACE
420 REM TIMES FOUND IN BUFFER AS ASCII DIGITS.
430 REM CONVERT TO A DECIMAL VALUE.
440 :
450 K=4 : GOSUB 900 : A=X : REM K = BUFFER OFFSET
460 K=12: GOSUB 900 : B=X
470 K=20: GOSUB 900 : C=X
480 P=49: G=F : H=F : REM F=G=H= ASCII ZERO
490 :
500 REM SORT WINNER, 2ND AND 3RD PLACE
510 :
520 IF A>B THEN F=F+1
530 IF A>C THEN F=F+1
540 IF B>A THEN G=G+1
550 IF B>C THEN G=G+1
560 IF C>A THEN H=H+1
570 IF C>B THEN H=H+1
REM PUT 1ST, 2ND, 3RD PLACEMENT IN BUFFER
600:
610 POKE 0,65 : POKE 1,45 : POKE 3,32
620 POKE 8,66 : POKE 9,45 : POKE11,32
630 POKE16,67 : POKE17,45 : POKE19,32
640 POKE 2,F : POKE10,G : POKE18,H
650:
660 REM NOW DISPLAY WINNERS IN LARGE LETTERS
670:
680 POKE 260,169 : X=USR(X)
690 PRINT CHR$(17);" LANE PLACE "; : REM HEADINGS
700:
710 REM SCREEN LOOKS LIKE THIS: LANE - PLACE - TIMER
720 REM
730 REM A-2 8034
740 REM B-1 7892
750 REM C-3 8047
760 REM
770:
780 REM WAIT FOR OPERATOR TO START NEXT RACE.
790 REM USER HITS RETURN TO CONTINUE.
800:
810 INPUT " press RETURN to race again ";A$:GOT0200
900:
910 REM CONVERT 4 ASCII CHARACTERS FROM BUFFER INTO NUMBER
920:
930 X=0:FOR I=K TO K+4:J=PEEK(I)-30:X=X*10+J:NEXT I:RETURN
11.05 IDEAL MACHINE LANGUAGE PROGRAM START

; PROGRAM NAME
;
; AUTHOR'S COPYRIGHT NOTICE
; PROGRAM DATE AND VERSION NUMBER
;
; PROGRAM DESCRIPTION
;
MONITOR EQU 0E003H ;EQUATE LABELS
;
ORG 0500H ;ORG VARIABLES
COUNT DEFB 0
;
ORG 0100H ;ORG PROGRAM

0100: C3 yy xx ENTER JP COLDST ;JUMP COLD START
0103: C3 yy xx WARM JP WARMST ;JUMP WARM START
0106: C3 yy xx SEND JP VIDEO ;OUTPUT DRIVER
0109: C3 yy xx RECEIVE JP KEYBD ;INPUT DRIVER
010C: C3 yy xx PRINTER JP OUTPUT ;PRINTER DRIVER
010F: C3 yy xx SOUND JP NOISE ;SOUND ROUTINES

; ETC
;
; HAVING THE PROGRAM START AT 100H MAKES
; IT EASY TO PLACE PROGRAM ON CP/M DISK.
;
; START PROGRAM WITH VECTOR JUMPS TO THE
; VARIOUS MAJOR ROUTINES USED BY THE
; PROGRAM. THUS, IF A USER MUST MODIFY
; THE PROGRAM TO SUIT HIS SYSTEM, HE
; ONLY HAS TO CHANGE THE JUMP ADDRESS TO
; ACCESS HIS NEW SUBSTITUTE ROUTINE.
;
0112: C0 BAUD DEFB 0COH ;SYSTEM PARAMETERS
0113: FE PORT DEFB 0FEH ;STORED HERE
;
; NOW PLACE A TABLE OF SYSTEM PARAMETERS
; USED SUCH AS BAUD RATE. A USER WHO
; USES A DIFFERENT PARAMETER NOW ONLY
; HAS TO STORE THE DIFFERENT VALUE HERE.
;
0114: MAIN ????? ?? ;BEGIN MAIN PROGRAM

ETC.
CHAPTER 12 --- I/O DRIVERS

12.01 RS232 OUTPUT DRIVER ROUTINE

TO DIRECT ALL OUTPUT TO BOTH THE RS232 PORT AND TO THE VIDEO, USE THE MONITOR COMMAND:

`>SE O=xxyy`

WHERE `xxyy` IS THE ENTRY ADDRESS FOR WHEREEVER THE ROUTINE IS LOCATED IN MEMORY. FOR THE ABOVE ADDRESSING THE COMMAND WOULD BE `>SE O=0000`. TO RESTORE THE OUTPUT TO JUST THE VIDEO DRIVER USE:

`>SE O=V`

THE ABOVE RS232 DRIVER IS FULLY RELOCATABLE.

12.02 RS232 DRIVER WITH PERFORATION SKIP.

THE ROUTINE WILL COUNT THE NUMBER OF LINES PRINTED AND AUTOMATICALLY ISSUE A FORM FEED AFTER 54 LINES OF PRINT.
SYNCHRONIZE THE LINE COUNTER WITH PAPER TOP-OF-FORM BY ENTERING A 0 INTO BYTE FFFF HEX.

EXAMPLE: >EN FFFF FROM THE MONITOR.
       FFFF: 0 /
       OR
       POKE -1,0 FROM BASIC.

TO LIST BASIC PROGRAM USE: POKE -1,0:LIST

TO DIRECT ALL OUTPUT TO BOTH THE RS232 PORT AND TO THE VIDEO, USE THE MONITOR COMMAND:

>SE O=xxyy

WHERE xxxyy IS THE ENTRY ADDRESS FOR WHERE EVER THE ROUTINE IS LOCATED IN MEMORY. FOR THE ABOVE ADDRESSING THE COMMAND WOULD BE >SE O=0000.

TO RESTORE THE OUTPUT TO JUST THE VIDEO DRIVER USE:

>SE O=V

THE ABOVE RS232 DRIVER IS FULLY RELOCATABLE.

==============================================
12.03 RS232 INPUT DRIVER ROUTINE.

0000:  FD 7E 3D  ENTRY  LD  A,(IY+03DH) ;UART CONTROL
0003:  F6 80  OR  80H      ;SET BIT 7
0005:  D3 FE  OUT (0FEH),A ;TURN ON RS232
0007:  C3 0F E0  JP  0E00FH ;GET RS232 INPUT

TO RECEIVE ALL INPUT FROM THE RS232 PORT, USE THE MONITOR COMMAND:

>SE I=xxyy

WHERE xxxyy IS THE ENTRY ADDRESS FOR WHERE EVER THE ROUTINE IS LOCATED IN MEMORY. FOR THE ABOVE ADDRESSING THE COMMAND WOULD BE >SE I=0000.

ONE LOSES KEYBOARD CONTROL OF THE SORCERER WHEN THE INPUT IS SET TO THE ABOVE RS232 INPUT DRIVER.

PERHAPS A BETTER USE OF THE ROUTINE WOULD BE TO CALL IT FROM AN APPLICATION PROGRAM WITH THE INP() FUNCTION.

EXAMPLE:  100 POKE 318,195:POKE 320,0
       110 X=INP(0) : A$=CHR$(X)

THE INP() EXAMPLE USES AN ENTRY ADDRESS OF 0000 HEX.
A$ IS THE CHARACTER RECEIVED FROM THE RS232 ROUTINE.

THE ABOVE RS232 DRIVER IS FULLY RELOCATABLE.
12.04 DUMB TERMINAL ROUTINE

; GET CHARACTERS FROM KEYBOARD AND SEND TO RS232.
; CHARACTERS RECEIVED FROM RS232 SENT TO VIDEO SCREEN.
; RETURN TO CALLING PROGRAM IF 'RUN/STOP' KEY IS Pressed.

ORG 2470H

2470: 3E C0 DUMB LD A,OCOH ; BAUD - RS232 ON
2472: D3 FE OUT (OFEH),A ; TURN RS232 ON
2474: DB FE LOOP IN A,(OFEH) ; LOOK FOR 'R/S'
2476: CB 47 BIT 0,A
2478: C8 RET Z ; ABORT IF 'R/S'
2479: CD 1C EB CALL 0EB1CH ; SCAN KEYBOARD
247C: 28 0A JR Z,SCAN-$ ; SKIP IF NO INPUT
247E: F5 PUSH AF
247F: DB FD UART IN A,(0FDH) ; WAIT UART DONE
2481: CB 47 BIT 0,A
2483: 28 FA JR Z,UART-$
2485: F1 POP AF
2486: D3 FC OUT (0FCH),A ; SEND CHARACTER
2488: DB FD SCAN IN A,(0FDH) ; CHAR RECEIVED
248A: CB 4F BIT 1,A
248C: 28 E6 JR Z,LOOP -$ ; LOOP IF NOTHING
248E: DB FC IN A,(0FCH) ; GET INCOMING CHAR
2490: CD 1B E0 CALL 0E01BH ; SEND TO VIDEO
2493: 18 DF JR LOOP- $ ; LOOP

12.05 VARIABLE LINE LENGTHS FOR PRINTERS.

100 POKE 322,J

J = LINE LENGTH FROM 0 TO 255.
BASIC WILL ISSUE A <CR> WHEN THE LINE LENGTH IS EXCEEDED.

EXAMPLE: 100 POKE 322,132 : REM 132 CHARACTER LINE

12.06 CENTRONICS SCREEN PRINT ROUTINE

100 POKE 262,195 : POKE 264,233
110 AD = -3968
120 FOR R = 0 TO 29 : FOR C = 0 TO 63
130 OUT 147,PEEK(AD) : AD = AD + 1
140 NEXT C
150 OUT 147,13 : REM SEND <CR> AT END-OF-LINE
160 NEXT R

THE POKE STATEMENTS ON LINE 100 INITIALIZE THE ROUTINE.
EACH 'OUT 147,##' SENDS ONE ASCII CHARACTER TO THE MONITOR'S CENTRONICS DRIVER ROUTINE.
12.07 CENTRONICS PRINTER DRIVER

100 OUT 255,J OR 128
110 OUT 255,J
120 OUT 255,J OR 128

These three OUT statements send the ASCII character value in J, and strobe the handshake on bit 7. This method of sending characters to a Centronics printer is much slower than the method of 12.06 where the 'OUT' command passes the ASCII directly to the monitor's Centronics routine.

12.08 ACCESS CENTRONICS PRINTER DRIVER FROM BASIC.

100 RAMSIZE = PEEK(-4095) * 256 + PEEK(-4096)
110 IF RAMSIZE > 32767 THEN RAMSIZE = RAMSIZE - 65536
120 :
130 POKE RAMSIZE - 47,147 : REM TURN PRINTER ON
140 :
200 POKE RAMSIZE - 47,240 : REM TURN PRINTER OFF

The above is equivalent to >SE O=L AND >SE O=V.

12.09 PROGRAMMING THE UART FOR PARITY OPTIONS

The UART used for serial transmission can be programmed for various parity and stop bit options.

0000: 3E xx START LD A,VALUE ;CONTROL PARAMETER
0002: D3 FD OUT (0FDH),A ;PROGRAM UART
0004: C9 RET ;BACK TO MAIN PROGRAM

FORMAT= X X X P PS S NB2 NB1 (8 BIT POSITIONS)

X = DON'T CARE BIT POSITION
P = PARITY ENABLE : 0=ENABLED 1=NULL
PS = PARITY SELECT : 0=ODD 1=EVEN
S = # OF STOP BITS: 0=ONE 1=THREE BITS
NB2 - NB1 = NUMBER OF BITS PER CHARACTER

<table>
<thead>
<tr>
<th>NB2</th>
<th>NB1</th>
<th># OF BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>

Example: VALUE = 0EH = 00001110 BINARY

7 BITS PER CHARACTER
2 STOP BITS
PARITY ENABLED
PARITY EVEN
CHAPTER 13 --- CASSETTE TAPE

13.01 WRITE DATA TO CASSETTE TAPE

```basic
100 MS=256*PEEK(-4095)+PEEK(-4096) : REM MEMORY SIZE
110 IF MS>32767 THEN MS=MS-65536
120 POKE MS-45,8 : POKE MS-44,1 : REM DISABLE KEYBOARD
130 POKE MS-41,16 : REM MOTOR #1 CONTROL
140 OUT 254,16 : REM TURN ON MOTOR #1.
150 POKE MS-47,18:POKE MS-46,244: REM OUTPUT TO TAPE
160 FOR J=1 TO 10
170 FOR K=1 TO 100:NEXT K : REM DELAY BETWEEN DATA
180 PRINT A$(J);"",";A(J) : REM PRINT DATA ON TAPE
190 NEXT J
200 POKE MS-47,27 : REM RESTORE VIDEO
210 OUT 254,0 : REM OFF MOTOR #1
220 POKE MS-45,24:POKE MS-44,224: REM RESTORE KEYBOARD
```

Both strings and numbers may be printed on the tape. "","" separates the data to match the input statement. The file created has no name and no CRC error checking. Keyboard is disabled so that its usual scan does not turn the cassette motors off.

13.02 READ DATA FROM CASSETTE TAPE

```basic
400 MS=256*PEEK(-4095)+PEEK(-4096) : REM MEMORY SIZE
410 IF MS>32767 THEN MS=MS-65536
420 POKE MS-45,15:POKE MS-44,224: REM TAPE TO INPUT
430 POKE MS-41,16 : REM MOTOR #1 CONTROL
440 OUT 254,16 : REM TURN ON MOTOR #1.
450 FOR J=1 TO 10
460 FOR K=1 TO 30:NEXT K : REM DELAY < THAN WHEN WRITTEN
470 INPUT A$(J),A(J) : REM INPUT DATA FROM TAPE
480 NEXT J
490 POKE MS-45,24 : REM RESTORE INPUT TO KEYBOARD
500 OUT 254,0 : REM TURN OFF MOTOR #1
```

This routine reads the data tapes created by 13.01. The input statement on line 470 is a matched statement to the print statement on line 180. The input vector points to the tape input routine rather than to the keyboard.
CHAPTER 14 --- EDITOR FOR BASIC

14.01 EDITOR FOR BASIC INSTRUCTIONS.

TO USE --
  >LO EDIT (Load the editor from 14.02)
  >SE I=7000 (For 32K version, ie. ORG address)
  >PP (Exit back to Basic)

Any cursor movement key, HOME, or TAB will activate edit mode
as indicated by the inverse video cursor.

CTRL E -- Expand the line by moving the rightmost characters
one space right for insertion ahead of the cursor.

CTRL R -- Reduce the line by deleting the character under the
cursor. Rightmost characters move one space left.

RUBOUT -- Rubout the character under the cursor and replace
with a space. Note that RUBOUT is now unshifted.

CTRL N -- Renumber all program statements in increments of 10
starting at 100. Starting line number is stored in
bytes 7136 and 7137 hex, and the increment is
stored in bytes 71A1 and 71F9 hex if you desire to
change them.

Example:  >EN 7136 (Enter Monitor by typing BYE)
            7136: E8 03 / (Change starting line # to 1000)
            >EN 71A1
            71A1: 64 / (Changes increment to 100)
            >EN 71F9
            71F9: 64 /
            >PP (Exit back to Basic)

CTRL U -- Up (ie. revive) a program lost due to mistakenly
typing NEW or CLOAD. (Hard reset destroys a
portion of your program near the start. However,
you may be able to revive from hard reset by using
CTRL U, listing the program, and deleting the line
number that is messed up. Loosing a couple lines
at the start of a program from a hard reset is
better than loosing the entire program.)

TAB -- Tab to the preset tabs in columns 1, 9, 17, 25, 33,
      41, 49, 57, and enter edit mode.

INVERSE CURSOR -- The editor's cursor is the inverse video of
the character it is sitting on top of.

LINE NUMBER EDITING -- Editing a statement's line number will
COPY the line under the new line number. The
original line still exits and may deleted, if you
desire, by typing the old line number and RETURN.
WRITING CODE -- Preferably enter edit mode before starting a statement. (If you enter the edit mode DURING the typing of a line, all of the text on the screen will be added behind what you have already typed. Therefore, return to the line in edit mode and hit RETURN a second time. Basic will then have the line as it appears on the screen.)

TRANSPARENCY -- When not in edit mode, the keyboard functions normally wherein you may RUN, CSAVE, LIST, write code, insert or delete lines, etc. The Monitor's \texttt{>SE I=K} will restore the regular Sorcerer keyboard routine.

CLEAR -- Using CLEAR will no longer generate a SYNTAX ERROR message.

CTRL X -- Cross reference the Basic program's variables and statement references such as GOTO, GOSUB, RESTORE, etc. This feature is present ONLY if you bought our CROSS REFERENCE program and added it to your Editor for Basic.
14.02 EDITOR FOR BASIC SOURCE LISTING

ORG 7000H

7000: CD 1C EB EDITOR CALL 0EB1CH ;SCAN KEYBOARD
7003: C8 EDITOR RET Z ;RETURN IF NOTHING
7004: C5 CALL OEB1CH ;SCAN KEYBOARD
7005: D5 PUSH BC
7006: E5 PUSH DE
7007: FD E5 PUSH HL
7009: CD A2 E1 CALL 0E1A2H ;GET IY
700C: FE 0C PUSH BC
700E: 20 05 JR NZ, L7015-$ ;CLEAR VIDEO CRT
7010: CD 0C E0 CALL 0E00CH ;RET WITH NOTHING
7013: 18 71 JR CLEAR-$ ;RET WITH NOTHING
7015: FE 01 L7015 CP 001H ;CURSOR LEFT
7017: 28 46 JR Z, VIDEO-$ ;CURSOR UP
7019: FE 11 CP 011H ;CURSOR HOME
701B: 28 42 JR Z, VIDEO-$ ;CURSOR RIGHT
701D: FE 13 CP 013H ;CURSOR UP
701F: 28 3E JR Z, VIDEO-$ ;CURSOR DOWN
7021: FE 0E CP 00BH ;TAB
7023: 28 3A JR Z, TAB-$ ;CTRL E - EXPAND
7025: FE 0A CP 005H ;CTRL E - EXPAND
7027: 28 36 JR Z, VIDEO-$ ;CTRL R -REDUCE
7029: FE 0B CP 00AH ;CTRL R - REDUCE
702B: 28 62 JR Z, TAB-$ ;CTRL N - RENUMBER
702D: FE 05 CP 005H ;CTRL N - RENUMBER
702F: 28 73 JR Z, EXPAND-$ ;CTRL R - REDUCE
7031: FE 12 CP 012H ;CTRL R - REDUCE
7033: 28 7C JR Z, REDUCE-$ ;CTRL U -UP PROG
7035: FE 0E CP 00EH ;CTRL U - UP PROG
7037: CA 35 71 JP Z, RENUM
7039: CA 31 71 JP Z, CLOAD
703B: FE 7F CP 07FH ;RUBOUT
703D: CA 47 71 LD A, 07FH ;CHANGE TO RUBOUT
703F: 18 06 JR L704D-$ ;CHANGE TO RUBOUT
7041: 20 04 JR NZ, L7047-$ ;CHANGE TO UNDERSC
7043: 3E 5F LD A, 05FH ;CHANGE TO UNDERSC
7045: 18 06 JR L704D-$ ;CHANGE TO UNDERSC
7047: FE 5F L7047 CP 05FH ;UNDERSCORE
7049: 20 02 JR NZ, L7047-$ ;UNDERSCORE
704B: 3E 7F LD A, 07FH ;CHANGE TO RUBOUT
704D: 47 L704D LD B, A
704F: 3A 18 71 LD A, (FLAG)
7051: B7 OR A
7053: 78 LD A, B ;RESTORE CHAR
7055: 28 32 JR Z, END-$ ;EXIT NORMAL MODE
7057: FE 0D CP 00DH ;<CR>
7059: FE 7F JR Z, LINE-$ ;RUBOUT
705B: 20 02 CP 07FH ;RUBOUT
705D: 3E 08 JR NZ, VIDEO-$ ;BACKSPACE
705F: 32 18 71 VIDEO LD (FLAG), A ;ENTER EDIT MODE
7062: CD 0C E0 CALL 0E00CH ;CHAR TO VIDEO
7065: CD 6D E9 CALL 0E9D6H ;GET CURSOR ADDR
; REVERSE VIDEO CHARACTER UNDER CURSOR

; 7068: 36 FE INVERSE LD (HL),0FEH ;INVERSE VIDEO
706A: FD 6E 67 LD L,(IY+067H) ;CHAR UNDER CURSOR
706D: 26 00 LD H,000H
706F: 29 ADD HL,HL
7070: 29 ADD HL,HL
7071: 29 ADD HL,HL ;ASCII OFFSET
7072: 11 00 F8 LD DE,0F800H ;TABLE BASE
7075: 19 ADD HL,DE ;HL = BIT PATTERN
7076: DD 21 F0 FF LD IX,OFFFOH ;GRAPHIC CHAR 254
707A: 06 08 LD B,008H ;ROW COUNTER
707C: 7E L707C LD A,(HL) ;GET BIT PATTERN
707D: 2F CPL ;INVERSE
707E: DD 77 00 LD (IX+000H),A ;CREATE GRAPHICS
7081: 23 INC HL ;NEXT ROW
7082: DD 23 INC IX
7084: 10 F6 DJNZ L707C-$ ;LOOP 8 TIMES

; 7086: AF CLEAR XOR A ;RETURN NOTHING
7087: FE 00 END CP 000H ;SET Z FLAG
7089: PD E1 POP IY
708B: E1 POP HL
708C: D1 POP DE
708D: C1 POP BC
708E: C9 RET ;EXIT

; PERFORM TAB FUNCTION

; 708F: CD E8 E9 TAB CALL 0E9E8H ;REPLACE CURSOR
7092: PD 7E 6A LD A,(IY+06AH) ;COLUMN NUMBER
7095: C6 08 ADD A,008H
7097: E6 38 AND 038H ;UNIFORM TABS
7099: PD 77 6A LD (IY+06AH),A ;NEW COLUMN
709C: 32 18 71 L709C LD (FLAG),A ;ENTER EDIT MODE
709F: CD CC E9 CALL 0E9CCH ;MOVE CURSOR
70A2: 18 C4 JR INVERSE-$ ;INVERSE VIDEO

; EXPAND OR REDUCE LINE ONE CHARACTER AT CURSOR LOCATION

; 70A4: CD C4 70 EXPAND CALL DELTA ;GET SCREEN ADDRS
70A7: 28 13 JR Z,L70BC-$
70A9: E5 PUSH HL
70AA: D1 POP DE
70AB: 2B DEC HL
70AC: ED B8 LDDR ;SHIFT LINE RIGHT
70AE: 23 INC HL
70AF: 18 0B JR L70BC-$
; REDUCE CALL DELTA ;GET SCREEN ADDRS
70B1: CD C4 70 JR Z,L70BC-$_$ 70B4: 28 06 CALL DELTA ;SHIFT LINE LEFT
70B6: D5 PUSH DE 70B9: ED B0 DEC HL
70B7: E1 POP HL 70BB: 2B LDIR ;INSERT SPACE
70B8: 23 INC HL 70BC: 3E 20 CALL O9E8H ;SAVE CURSOR
70B9: ED B0 L70BC LD A,020H
70BB: 2B DEC HL 70BF: CD CC E9 JR INVERSE-$_$
70BC: 3E 20 L70BC LD (HL),A
70BE: 77 LD A,020H ;SAVE CURSOR
70BF: CD CC E9 CALL O9E8H ;SAVE CURSOR

; CALCULATE NUMBER OF CHARACTERS TO SHIFT
; ON EXIT: DE = PRESENT CURSOR ADDR. HL = LINE END ADDR.
70C2: 18 A4 CALL O9E8H ;REPLACE CURSOR
70C4: CD E8 E9 DELTA CALL O9E8H ;REPLACE CURSOR
70C7: 7D LD A,L
70C8: E6 3F AND 03F
70CA: D6 40 SUB 040H
70CC: 2F CPL
70CD: 4F LD C,A
70CE: 06 00 LD B,000H ;BC = # CHAR
70D0: EB EX DE,HL ;DE = PRESENT
70D1: C5 PUSH BC
70D2: E1 POP HL
70D3: 19 ADD HL,DE ;HL = LINE END
70D4: B1 OR C ;TEST BC=0
70D5: C9 RET
70D6: 21 F4 70 LINE LD HL,NEW ;NEW INPUT VECTOR
70D9: FD 75 41 LD (IY+041H),L
70DC: FD 74 42 LD (IY+042H),H
70DF: CD CC 70 CALL DELTA ;GET LINE END ADDR
70E2: 06 41 LD B,041H ;65 CHAR DEFAULT
70E4: 3E 20 LD A,020H
70E6: BE L70E6 CP (HL)
70E7: 20 04 JR NZ,L70ED-$_$
70E9: 2B DEC HL
70EA: 10 FA DJNZ L70E6-$_$ ;SUBTRACT SPACES
70EC: 04 INC B ;B <> 0
70ED: 78 L70ED LD A,B ;# CHAR TO MOVE
70EE: FD 36 6A 00 LD (IY+06AH),000H ;FIRST COLUMN
70F2: 18 A8 JR L709C-$_$ ;GO UPDATE MEMORY
I/O INPUT FROM SCREEN TO BASIC. REPLACES KEYBOARD INPUT.

70F4: FD E5 NEW PUSH IY
70F6: CD A2 E1 CALL 0E1A2H ;GET IY
70F9: 3A 18 71 LD A, (FLAG) ;DEC CHAR COUNTER
70FC: 3D DEC A
70FD: 32 18 71 LD (FLAG), A
7100: 20 0F JR NZ, L7111- $ ;LOOP TIL DONE
7102: E5 PUSH HL
7103: 21 00 70 LD HL, EDITOR ;RESTORE KEYBOARD
7106: FD 75 41 LD (IY + 041H), L
7109: FD 74 42 LD (IY + 042H), H
710C: E1 POP HL
710D: 3E 0D LD A, 00DH ;RETURN WITH <CR>
710F: 18 03 JR L7114- $
7111: FD 7E 67 L7111 LD A, (IY + 067H) ;GET CHARACTER
7114: B7 L7114 OR A ;SET NZ FLAG
7115: FD E1 POP IY
7117: C9 RET

7118: 00 FLAG DEFB 0

; RESTORE PROGRAM AFTER CLOAD

7119: 21 D9 01 CLOAD LD HL, 001D9H ;PROGRAM START
711C: AF XOR A
711D: BE L711D CP (HL) ;LOOK FOR ZERO
711E: 23 INC HL
711F: 20 FC JR NZ, L711D- $ ;LOOP TIL FOUND
7121: 22 D5 01 LD (L01D5), HL ;RESTORE POINTER
7124: 5E L7124 LD E, (HL) ;NEXT LINE ADDR
7125: 23 INC HL
7126: 56 LD D, (HL)
7127: EB EX DE, HL
7128: 7D LD A, L
7129: B4 OR H
712A: 20 F8 JR NZ, L7124- $ ;FIND PROG END
712C: 13 INC DE
712D: 13 INC DE
712E: ED 53 B7 01 LD (L01B7), DE ;SAVE END ADDRESS
7132: C3 86 70 JP CLEAR

; PERFORM RENUMBER FUNCTION

7135: 21 64 00 RENUM LD HL, 100 ;DEFAULT START
7138: 22 58 72 LD (INIT), HL
713B: 21 D5 01 LD HL, 01D5H ;PROG START ADDR
713E: 22 54 72 LD (NEXT), HL
; PUT ADDRESS OF NEXT LINE IN (NEXT)
;
7141: 2A 54 72 NEXTLIN LD HL,(NEXT)
7144: 22 4E 72 LD (PRESENT),HL
7147: 5E LD E,(HL)
7148: 23 INC HL
7149: 56 LD D,(HL) ;DE = NEXT LINE
714A: 23 INC HL
714B: ED 53 54 72 LD (NEXT),DE
714F: 7B LD A,E
7150: B2 OR D
7151: CA E6 71 JP Z,PASS2 ;TEST END OF PROG
7154: 23 INC HL
7155: 23 INC HL
;
; LOOK FOR REM, GOTO, GOSUB, THEN
;
7156: 7E LOOK LD A,(HL) ;GET CHARACTER
7157: 23 INC HL
7158: FE 00 CP 000H ;END OF LINE?
L7158
715C: 28 E5 JR Z,NEXTLIN-$
715E: 28 E1 CP 089H ;REM ?
7160: FE 89 CP 08DH ;GOTO ?
7162: 28 0E JR Z,GOTO-$
7164: FE 8D CP 08CH ;RESTORE ?
7166: 28 0A JR Z,GOTO-$
7168: FE 8C CP 0A2H ;THEN
716A: 28 06 JR Z,GOTO-$
716C: FE A2 CP
716E: 28 02 JR Z,GOTO-$
7170: 18 E4 JR LOOK-$ ;CONTINUE SEARCH
;
; PROCESS GOTO, GOSUB, THEN STATEMENTS
;
7172: E5 GOTO PUSH HL ;SAVE BEGIN ADDR
7173: CD EA C7 CALL 0C7EAH ;CONVERT TO BINARY
7176: 22 50 72 LD (POINT),HL ;END OF # ADDR
7179: AF XOR A
717A: C1 POP BC
717B: ED 42 SBC HL,BC ;LENGTH OF LINE #
717D: 22 56 72 LD (LENGTH),HL
; SEARCH FOR LINE # MATCH
;
7180: 2A 58 72  LD  HL,(INIT)
7183: 22 52 72  LD  (COUNT),HL ;INITIALIZE LINE #
7186: 21 D5 01  LD  HL,01D5H
7189: 4E       L7189
718A: 23  INC  HL
718B: 46  LD  B,(HL) ;NEXT LINE ADDR
718C: 23  INC  HL
718D: 79  LD  A,C
718E: B0  OR  B
718F: 28 48  JR  Z,L71D9-$ ;END OF PROG?
7191: D5  PUSH DE  ;SAVE MATCH #
7192: 5E  LD  E,(HL)
7193: 23  INC  HL
7194: 56  LD  D,(HL) ;CURRENT LINE #
7195: EB  EX DE,HL
7196: D1  POP  DE
7197: CD 74 C5  CALL OC574H ;COMPARE HL-DE
719A: 28 10  JR  Z,L71AC-$
719C: D5  PUSH DE  ;NO MATCH
719D: 2A 52 72  LD  HL,(COUNT)
71A0: 11 0A 00  LD  DE,10 ;ADD 10 TO COUNT
71A3: 19  ADD HL,DE
71A4: 22 52 72  LD  (COUNT),HL
71A7: D1  POP  DE
71A8: 69  LD  L,C
71A9: 60  LD  H,B ;NEXT LINE ADDR
71AA: 18 DD  JR  L7189-$
;
; FOUND MATCH, CONVERT LINE # TO DECIMAL AND REPLACE
;
71AC: CD B2 C9  L71AC  CALL OC9B2H ;FIRST COLUMN
71AF: 2A 52 72  LD  HL,(COUNT) ;# TO CONVERT
71B2: CD BB D7  CALL 0D7BBH ;PRINT DECIMAL #
71B5: ED 4B 56 72  LD  BC,(LENGTH) ;LENGTH OF #
71B9: 41  LD  B,C
71BA: CD A2 E1  CALL 0E1A2H ;GET IY
71BD: CD D6 E9  CALL 0E9D6H ;GET CURSOR ADDR
71C0: 7D  LD  A,L
71C1: E6 07  AND 007H
71C3: 4F  LD  C,A
71C4: 78  LD  A,B
71C5: 91  SUB C
71C6: FC 0C 72  CALL M,EXPAND ;MOVE PROG DOWN
71C9: 2B  DEC HL
71CA: DD 2A 50 72  LD  IX,(POINT)
71CE: DD 2B  DEC IX
;
; MOVE NUMBER ON SCREEN TO MEMORY IN REVERSE ORDER
;
71D0: 7E       L71D0  LD  A,(HL)
71D1: DD 77 00  LD  (IX+000H),A ;TRANSFER CHAR
71D4: 2B  DEC HL
71D5: DD 2B  DEC IX
71D7: 10 F7  DJNZ L71D0-$
; CHECK FOR COMMA AFTER NUMBER
;
L71D9 LD HL,(POINT) ;LAST LOOK ADDR
L71D9 LD A,(HL)
CP 02CH ;IS IT A COMMA?
INC HL
JP Z,GOTO ;YES, CONVERT #
JP L7158 ;NO, CONTINUE SCAN
;
; PASS TWO - RENUMBER ALL LINES
;
L71E6: 21 D5 01 PASS2 LD HL,01D5H ;START OF PROGRAM
L71E9: ED 5B 58 72 LD DE,(INIT) ;NEW LINE NUMBER
L71ED: 4E L71ED LD C,(HL)
INC HL
LD B,(HL) ;GET FORWARD LINK
INC HL
LD A,C
OR B
JR Z,L7201-$ ;END OF PROGRAM?
L71F5: 73 LD (HL),E
L71F6: 23 INC HL
L71F7: 72 LD (HL),D ;STORE NEW LINE #
L71F8: 21 0A 00 LD HL,10 ;DEFAULT INCREMENT
L71FB: 19 ADD HL,DE
L71FC: EB EX DE,HL
L71FD: 69 LD L,C
L71FE: 60 LD H,B
L71FF: 18 EC JR L71ED-$ ;LOOP TIL PROG END
;
L7201: 21 AF C2 L7201 LD HL,0C2AFH
L7204: CD 15 D0 CALL 0D015H ;PRINT READY
L7207: 3E 0D LD A,00DH ;RETURN <CR>
L7209: C3 87 70 JP END
;
; MOVE PROGRAM DOWN
;
L720C: C5 EXPAND PUSH BC
L720D: E5 PUSH HL
L720E: 79 LD A,C
L720F: 90 SUB B ;A = # TO MOVE
L7210: 4F LD C,A
L7211: 06 00 LD B,000H
L7213: C5 PUSH BC
L7214: 2A 54 72 LD HL,(NEXT)
L7217: 09 ADD HL,BC
L7218: 22 54 72 LD (NEXT),HL ;ADJUST POINTERS
L721B: 2A 50 72 LD HL,(POINT)
L721E: 09 ADD HL,BC
L721F: 22 50 72 LD (POINT),HL ;ADJUST POINTERS
L7222: EB EX DE,HL
L7223: 2A B7 01 LD HL,(01B7H) ;NEW PROG END ADDR
7226: E5
7227: 09
7228: 22 B7 01
722B: E5
722C: ED 52
722E: E5
722F: C1
7230: D1
7231: E1
7232: 03
7233: ED B8
7235: 2A 4E 72
7238: C1
7239: 5E
723A: 23
723B: 56
723C: 7B
723D: B2
723E: 28 0A
7240: 2B
7241: EB
7242: 09
7243: EB
7244: 73
7245: 23
7246: 72
7247: EB
7248: 18 EF
724A: E1
724B: C1
724C: 41
724D: 09
724E: 00
7250: 00
7252: 00
7254: 00
7256: 00
7258: 00

PUSH HL
ADD HL,BC
LD (01B7H),HL
PUSH HL
SBC HL,DE
PUSH HL
POP BC
POP DE
POP HL
INC BC
LDDR ;MOVE PROG DOWN
LD HL,(PRESENT)
POP BC ;DISTANCE MOVED
RELINK
LD E,(HL) ;RELINK BOTTOM
INC HL
LD D,(HL)
LD A,E
OR D
JR Z,L724A-$ ;AT PROG END?
DEC HL
EX DE,HL
ADD HL,BC
EX DE,HL
LD (HL),E
INC HL
LD (HL),D ;ADJUST LINK
EX DE,HL
JR RELINK-$
L724A POP HL
POP BC
LD B,C ;RELINK FINISHED
PRESENDEFW 0
POIN TDEFW 0
COUNTDEFW 0
NEXTDEFW 0
LENGTHDEFW 0
INITDEFW 0
CHAPTER 15 --- CP/M

15.01 SAVE ROMPAC BASIC PROGRAM ON CP/M DISK.

REQUIREMENT: DISK BOOT ADDRESS MUST NOT CONFLICT WITH THE ROMPAC ADDRESSES OF C000 - DFFF HEX.

CLOAD XMPLE
BYE
>DU 1B7 1B8
01B7: C4 24
>GO B900
A>SAVE 36 XMPLE.COM
A>XMPLE
READY
RUN

JUST CONVERT CONTENTS OF BYTE 1B8 TO DECIMAL FOR xx.
WHEN CP/M LOADS THE PROGRAM AND BEGINS EXECUTION, IT FINDS A JP 0C06BH AT 100H. THIS JUMP WAS PLACED AT 100H BY THE BASIC ROMPAC.

15.02 SAVE WORD PROCESSOR ROMPAC FILES ON CP/M DISK.

REQUIREMENT: DISK BOOT ADDRESS MUST NOT CONFLICT WITH THE ROMPAC ADDRESSES OF C000 - DFFF HEX.

COMMAND> X
>DU 74A 74B
074A: 45 1A
>EN 100
0100: C3 03 CO
>GO B900
A>SAVE 26 WPFILE.COM
A>WPFILE

EXIT TO MONITOR FROM WP ROMPAC.
FIND END OF WORD PROCESSOR FILE.
TYPICAL END-OF-FILE ADDRESS
NEED JUMP TO ROMPAC WARM START.
TYPICAL DISK BOOT ADDRESS.
SAVE xx BLOCKS THROUGH FILE END.
26 DECIMAL = 1A HEX FROM 74B.
RELOAD AND JUMP TO WARM START.
15.03 CP/M COMMANDS

**PIP**

- **PIP A:=B:XXX.COM**  
  Copy file XXX from B to A
- **PIP A:=B:*.***  
  Copy all files from B to A
- **PIP B:=C:*.*.COM**  
  Copy all COM files from C to B
- **PIP B:PROG.BAK=C:PROG.COM**  
  Copy backup of PROG from C to B
- **PIP B:=C:*.*.COM[V]**  
  Copy all COM files from C to B with verification
- **PIP B:XXX=A:YYY**  
  Copy file X from A to B and rename XXX
- **PIP B:=A:XXX.***  
  Copy all files with name of XXX on A onto B

**REN**

- **REN NEW.COM=OLD.COM**  
  Rename file name
- **REN NEW.BAK=OLD.COM**  
  Rename file OLD type COM to NEW type BAK
- **REN XXX.BAK=XXX.COM**  
  Rename filetype
- **REN C:XXX.COM=YYY.COM**  
  Rename file YYY on C to XXX

**ERA**

- **ERA XXX.COM**  
  Erase file XXX with type COM
- **ERA *.*.DAT**  
  Erase all file types of DAT
- **ERA XXX.***  
  Erase all files with name of XXX
- **ERA *.***  
  Erase all files
- **ERA C:*.*.COM**  
  Erase all COM type files on C

**SAVE**

- **A>SAVE xx PROGNAME.COM**  
  Save file on A
- **A>SAVE xx B:PROGNAME.COM**  
  Save file on B

**CTRL CHARACTERS**

- **CTRL X** - delete line typed
- **CTRL R** - retype line
- **CTRL C** - reboot system (warm)
15.04 BIOS MODIFICATIONS TO GIVE BACKSPACE TYPE RUBOUTS

CONSOLE INPUT

This routine must get a character from the console, and return the character in the ACCUMULATOR.

If the character is a RUBOUT, then a new delete flag is set to cause the backspace sequence of characters to be sent to the VIDEO display by the CONSOLE OUTPUT driver.

B24D: CD 26 B2 CINP CALL CSTAT ; IS A CHARACTER READY?
B250: CA 4D B2 JZ CINP ; NO--WAIT FOR IT.
B253: CD 09 E0 CALL KEYBRD ; GET THE CHARACTER.
B256: CA 4D B2 JZ CINP ; TRY AGAIN IF NOT IN TIME
B259: FE 0B CPI 'K' -40H ; IS THE CHARACTER A TAB?
B25B: CA 6E B2 JZ TABIT ; YES--CONVERT IT.
B25E: FE 09 CPI 'I' -40H ; IS IT A CONTROL I ?
B260: CA 71 B2 JZ UNTAB ; YES--
B263: FE 7F CPI 5FH ; IS THIS AN UNDERSCORE?
B265: C0 RNZ ; NO, FINISHED WITH CINP
B266: 3E 01 MVI A,1 ; SET
B268: 32 EA B2 STA DELF ; DELETE FLAG
B26B: 3E 7F MVI A,7FH ; CHANGE TO RUBOUT
B26D: C9 RET

B26E: 3E 09 TABIT MVI A,'I' -40H ; CHANGE TO THE CP/M TAB
B270: C9 RET

B271: 3E 0B UNTAB MVI A,'K' -40H ; CHANGE TO CONTROL K.
B273: C9 RET

CONSOLE OUTPUT

This routine will be called with the character to be output in the 'C' REGISTER. If the delete flag is set, then generate the characters to erase the deleted character from the video display rather than echo it.

B274: 3A EA B2 COUT LDA DELF ; CHECK
B277: FE 01 CPI 1 ; DELETE FLAG
B279: C2 97 B2 JNZ JMPRBO ; NO, GO JMPRBO
B27C: E5 PUSH H
B27D: 21 EB B2 LXI H,CHAR2
B280: 3E 01 MVI A,1 ; MOVE CURSOR LEFT
B282: CD 1B E0 CALL VIDEO
B285: 3E 20 MVI A,020H ; SPACE ERASE CHAR
B287: CD 1B E0 CALL VIDEO
B28A: 3E 01 MVI A,1 ; MOVE CURSOR LEFT
B28C: CD 1B E0 CALL VIDEO
B28F: 34  INR M  ;ADJUST CHAR COUNT
B290: E1  POP H
B291: 3E 00  MVI A,0  ;RESET
B293: 32 EA B2  STA DELF  ;DELETE FLAG
B296: C9  RET

B297: 79  JMPRBO MOV A,C  ;GET IT TO THE ACUM.
B298: E5  PUSH H  ;SAVE HL.
B299: 21 EB B2  LXI H,CHAR2  ;POINT TO CHAR COUNT
B29C: FE 0D  CPI ODH  ;CARRIAGE RETURN?
B29E: CA B5 B2  JZ NEW  ;FORM FEED?
B2A1: FE 0C  CPI OCH
B2A3: CA B5 B2  JZ NEW
B2A6: 35  DCR M  ;COUNT OFF CHAR.
B2A7: C2 B7 B2  JNZ VOUT  ;GO WORK.
B2AA: 3E 0D  MVI A,13
B2AC: CD 1B E0  CALL VIDEO  ;SEND CR
B2AF: 3E 0A  MVI A,10
B2B1: CD 1B E0  CALL VIDEO  ;SEND A LINE FEED
B2B5: 36 42  NEW MVI M,42H  ;RESET CHARACTER COUNT
B2B7: CD 1B E0  VOUT CALL VIDEO  ;MORE WORK.
B2BA: E1  POP H  ;JUST LIKE WE CAME IN
B2BB: C9  RET  ;BACK TO CALLER

B2EA: 00  DELF DEFB 0  ;DELETE FLAG
This program operates with the Development Pac by bringing a Word Processor File into the DEVPAC file area from CP/M disk. The program then sets the I/O vectors, and jumps to the Development Pac.

COPYRIGHT (C), Frank Root, Sept. 16, 1981

ORG 100H
BDOS EQU 0005H ;DOS ENTRY POINT
OPENF EQU 15 ;FILE OPEN
READF EQU 20 ;READ FUNCTION
DMAF EQU 26 ;SET DMA ADDRESS FUNCTION
FILE EQU 05A80H ;START ADDRESS OF FILE FOR
 ;46K MEMORY CONFIGURATION
FCB EQU 5CH ;FILE CONTROL BLOCK
FCBCR EQU FCB+32 ;CURRENT (NEXT) RECORD
BUFINC EQU 80H ;READ BLOCK LENGTH

OPEN
LD DE, FCB ;OPEN NAMED FILE
LD C, OPENF
CALL BDOS
CP 255 ;CHECK FOR OPEN ERROR
RET Z ;BAD OPEN
XOR A
LD (FCBCR), A ;SET FIRST RECORD TO 0

MAIN
PUSH BC ;SAVE BC
LD BC, FILE
LD (FILBUF), BC
CALL MOVBUF
CALL DISKR
CP 0 ;CHECK FOR ERROR
JR Z, LOOP-$
POP BC
RET ;BACK TO CPM ON READ ERROR

LOOP
PUSH HL ;SAVE HL
LD HL, (FILBUF)
LD BC, BUFINC ;BUFFER LENGTH
ADD HL, BC
LD (FILBUF), HL
POP HL
CALL MOVBUF
CALL DISKR
CP 0 ;CHECK FOR ERROR
JR Z, LOOP-$
POP BC
RET ;BACK TO CPM ON READ ERROR
DISKR       PUSH HL                ;READ ONE BUFFER FROM DISK
         PUSH DE
         PUSH BC
         LD   DE,FCB
         LD   C,READF
         CALL BDOS
         POP  BC
         POP  DE
         POP  HL
         CP   1            ;CHECK FOR EOF
         JR   Z,FINIS-$
         RET

MOVBUF      PUSH HL                ;MOVE BUFFER TO RAM
         PUSH DE
         PUSH BC
         LD   DE,(FILBUF)
         LD   C,DMAF
         CALL BDOS
         POP  BC
         POP  DE
         POP  HL
         RET

; CI EQU 0F01EH
CO EQU 0F020H
OI EQU 0F022H
OO EQU 0F024H
SI EQU 0F026H
SO EQU 0F028H
SK EQU OC547H
SV EQU OC54EH
AO EQU OC60AH
AI EQU OC5F5H
BO EQU OC624H
;
FINIS       LD   BC,SK            ;SET DEVELOPMENT PAC VECTORS
         LD   (CI),BC
         LD   BC,SV
         LD   (CO),BC
         LD   BC,CO
         LD   (OI),BC
         LD   BC,CI
         LD   (OO),BC
         LD   BC,BO
         LD   (SI),BC
         LD   BC,SV
         LD   (SO),BC
         LD   BC,FILE
;
ELOOP INC BC ;NOW PUT A '0' AT EOF
LD A,(BC)
CP 01AH ;IS IT DISK EOF MARKER?
JR NZ,ELOOP-$
XOR A
LD (BC),A
POP BC ;POP DISKR RETURN ADDRESS
POP BC ;RESTORE BC FROM MAIN PUSH
JP 0E000H ;TO RESET ADDRESS OF DEVPAC

FILBUF DEFW 0
;
END
CHAPTER 16 --- WORD PROCESSOR

16.01 WORD PROCESSOR PRINTER DRIVER

INSTALL YOUR OWN PRINTER DRIVER BY PUTTING THE ADDRESS OF YOUR ROUTINE IN BYTES 7E7 AND 7E8 HEX. USES DRIVER #1 IN THE 'Y' TABLE.

0000: 21 yy xx ENTRY LD HL,NEWDRIVER
0003: 22 E7 07 LD (07E7H),HL

------------------------------------------------------------------------------------------------------------------------------

16.02 SALVAGE WORD PROCESSOR FILE FROM RESET.

IF THE WORD PROCESSOR RESETS, AND YOU ARE CURSING BECAUSE YOUR FILE WASN'T SAVED ON TAPE OR DISK YET, ALL IS NOT LOST. YOU MAY BE ABLE TO SALVAGE YOUR FILE (EVERYTHING EXCEPT FOR THE FIRST 175 CHARACTERS.)

GO TO THE MONITOR AND TRY: >MO 800 8CE 801 >PP

YOUR FILE IS STILL IN MEMORY, AND ONLY THE FIRST FEW LINES OF TEXT HAVE BEEN LOST. THESE LOST LINES WILL HAVE TO BE REENTERED.

------------------------------------------------------------------------------------------------------------------------------

16.03 PRINTER DRIVER TO SEND ESCAPE SEQUENCES

THIS DRIVER PATCHES INTO THE WORD PROCESSOR ROMPAC PRINTER DRIVER TO ALLOW ONE TO SEND ESCAPE SEQUENCES TO ACCESS A PRINTER'S SPECIAL FEATURES SUCH AS FONT SIZES AND FORM FEED.

CREATE A TABLE OF ESCAPE SEQUENCES THAT YOUR PRINTER UNDERSTANDS AT THE END OF THIS ROUTINE. THE TABLE IS CONSTRUCTED WITH A MATCH LETTER FOLLOWED BY A FIVE BYTE ESCAPE SEQUENCE. THUS, THE TABLE IS A MULTIPLE OF 6 BYTES IN LENGTH.

THIS DRIVER WATCHES FOR AN @ SIGN IN THE WORD PROCESSOR TEXT TO SIGNAL THAT THE NEXT LETTER IS TO GENERATE SOME DESIRED ESCAPE SEQUENCE. NEITHER THE @ SIGN, NOR THE FOLLOWING MATCH LETTER WILL BE PRINTED. INSTEAD, THE ROUTINE WILL SEARCH THE TABLE FOR THIS LETTER AND SEND THE 5 BYTE SEQUENCE THAT FOLLOWS THE MATCH LETTER.

AFTER LOADING THIS ROUTINE, >GO 0 TO INITIALIZE THE ROUTINE. THE ROUTINE WILL PATCH ITSELF INTO THE WORD PROCESSOR PRINTER DRIVER. IN THE 'Y' TABLE, USE PRINTER ROUTINE #1.

; 0000: AF INITZ XOR A
0001: 32 61 00 LD (FLAG),A ;CLEAR FLAG
0004: 21 0D 00 LD HL,DRIVER
0007: 22 E7 07 LD (07E7H),HL ;WP VECTOR
000A: C3 FA DF JP ODFFAH ;GO WARM START
THE FOLLOWING DRIVER NOW SUBSTITUTES FOR THE REGULAR WORD PROCESSOR PRINTER ROUTINE.
WHEN THE @ SIGN IS SEEN, A FLAG IS SET TO INDICATE THAT THE NEXT CHARACTER SHOULD SELECT A SEQUENCE FROM THE TABLE.

AN @ SIGN WAS THE PREVIOUS CHARACTER. TAKE THE PRESENT CHARACTER AND SEARCH THE TABLE FOR A MATCH. IF WE ARRIVE AT THE BOTTOM OF THE TABLE WITHOUT A MATCH, THEN GO AHEAD AND PRINT THE CHARACTER. THIS ALLOWS THE @ SIGN TO BE PRINTED FROM OUR TEXT BY USING TWO @ SIGNS TOGETHER.

A SEQUENCE HAS BEEN LOCATED IN THE TABLE. NOW SEND OUT THE 5 BYTES THAT FOLLOW IN THE TABLE.
; USEFUL EQUATES
ESC  EQU  1BH
CR   EQU  0DH
LF   EQU  0AH
BELL EQU  07H
NULL EQU  00H
;
0061: 00  FLAG  DEFB 0
;
0062: 45  BASE  DEFB 'E'
0063: 1B  DEFB ESC
0064: 26  DEFB ')'
0065: 6B  DEFB 'k'
0066: 31  DEFB '1'
0067: 00  DEFB NULL
;
0068: 00  DEFB 0

OTHER SEQUENCES FOR YOUR SERIAL PRINTER WOULD CONTINUE AT
ADDRESS 0068H. JUST BE SURE TO MARK THE END OF THE TABLE WITH
A 0 BYTE IN THE POSITION WHERE A MATCH LETTER WOULD GO. ALSO,
PAD A SEQUENCE WITH 'NULL' FILLER CHARACTERS IF IT IS LESS THAN
FIVE BYTES IN LENGTH AS IN THE ABOVE EXAMPLE.

EXAMPLE TEXT:    @E ENTER EXPANDED MODE.
                @C ENTER COMPRESSED MODE.
                @N RETURN TO NORMAL SIZED PRINT.

THE @E, @N, AND @C DO NOT PRINT WHEN THE FEATURE IS ACCESSED.
THEY ARE SHOWN IN THE EXAMPLE TEXT AS IT WOULD APPEAR ON THE
VIDEO DISPLAY.
16.03 CORRECTION

; AN @ SIGN WAS THE PREVIOUS CHARACTER. TAKE THE PRESENT
; CHARACTER AND SEARCH THE TABLE FOR A MATCH. IF WE ARRIVE
; AT THE BOTTOM OF THE TABLE WITHOUT A MATCH, THEN GO
; AHEAD AND PRINT THE CHARACTER. THIS ALLOWS THE @ SIGN
; TO BE PRINTED FROM OUR TEXT BY USING TWO @ SIGNS TOGETHER.

0021: AF CRTL XOR A ;CLEAR FLAG
0022: 32 61 00 LD (FLAG),A
0025: 21 62 00 LD HL,BASE ;TABLE BASE
0028: 7E LOOP LD A,(HL) ;GET LETTER
0029: B7 OR A ;TEST 0 END
002A: 28 0B JR Z, METO0-$ ;NOT IN TABLE
002C: B9 CP C ;MATCH CHAR?
002D: 28 10 JR Z, FOUND-$ ;GO SEND SEQ.
002F: 23 INC HL ;SKIP TO NEXT
0030: 23 INC HL ;ENTRY IN
0031: 23 INC HL ;TABLE WHEN
0032: 23 INC HL ;THERE IS NO
0033: 23 INC HL ;MATCH.
0034: 23 INC HL ;SEQ IS 6 LONG
0035: 18 F1 JR LOOP-$

SEND THE CHARACTER TO THE WORD PROCESSOR PRINTER DRIVER.

0037: 79 MET00 LD A,C ;GET CHARACTER
0038: CD 90 DE PRINT CALL 0DE90H ;WP SERIAL
003B: E1 EXIT POP HL
003C: C1 POP BC
003D: F1 POP AF ;RESTORE REGS
003E: C9 RET ;BACK TO WP

; A SEQUENCE HAS BEEN LOCATED IN THE TABLE. NOW SEND OUT THE
; 5 BYTES THAT FOLLOW IN THE TABLE.

003F: 06 05 FOUND LD B,005H ;SEND 5 BYTES
0041: 23 INC HL ;FIRST BYTE.
0042: 7E LOOP2 LD A, (HL)
0043: CD 4B 00 CALL SERIAL ;SERIAL OUT
0046: 23 INC HL ;NEXT BYTE
0047: 10 F9 DJNZ LOOP2-$ ;LOOP THRU 5
0049: 18 F0 JR EXIT-$ ;DONE NOW

; A SERIAL PRINTER ROUTINE FOR 1200 BAUD OR 300 BAUD.
; THIS ROUTINE IS USED TO SEND OUT THE SEQUENCE SO THAT
; THE WORD PROCESSOR CANNOT HAVE CONTROL UNTIL WE ARE THROUGH.

004B: F5 SERIAL PUSH AF
004C: FD 7E 3D LD A, (IY+03DH) ;BAUD RATE
004F: F6 80 OR 080H ;RS232 ON
0051: D3 FE OUT (0FEH),A
0053: FD 77 45 LD (IY+045H),A ;TAPE STATUS
0055: F1 POP AF
0057: CD 12 E0 CALL LE012 ;SERIAL OUT
005A: DB FD LOOP3 IN A, (0FDH) ;WAIT FOR UART
005C: CB 47 BIT 0, A
005E: 28 FA JR Z, LOOP3-$
0060: C9 RET
CHAPTER 17  ---  DEVELOPMENT PAC

17.01 PAUSE DEVELOPMENT PAC LISTINGS

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000:CD</td>
<td>15 CO</td>
<td>ENTRY CALL QUICKCHECK</td>
</tr>
<tr>
<td>0003:20</td>
<td>FB</td>
<td>JR NZ,ENTRY-$</td>
</tr>
<tr>
<td>0005:7A</td>
<td></td>
<td>LD A,D</td>
</tr>
<tr>
<td>0006:CD</td>
<td>1B E0</td>
<td>CALL VIDEO</td>
</tr>
<tr>
<td>0009:FE</td>
<td>0A</td>
<td>CP LINEFEED</td>
</tr>
<tr>
<td>000B:C0</td>
<td></td>
<td>RET NZ</td>
</tr>
<tr>
<td>000C:C3</td>
<td>1B E0</td>
<td>JP VIDEO</td>
</tr>
</tbody>
</table>

UNDER DDT80, CHANGE THE :SO AND :CO VECTORS TO 0000 HEX. NOW, HOLDING DOWN THE RUN/STOP KEY WILL PAUSE A LISTING.

17.02 MEMORY PARTITIONS FOR 32K CONFIGURATION.

<table>
<thead>
<tr>
<th>Partition</th>
<th>Description</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>I/O AND STACK</td>
<td>7D00 - 7FFF</td>
</tr>
<tr>
<td>#2</td>
<td>PROGRAM SOURCE, 'B' BUFFER</td>
<td>3E80 - 7CFF</td>
</tr>
<tr>
<td>#3</td>
<td>ASSEMBLED CODE, 'A' BUFFER</td>
<td>1F40 - 3E7F</td>
</tr>
<tr>
<td>#4</td>
<td>USER LOAD AREA</td>
<td>013A - 1F3F</td>
</tr>
<tr>
<td>#5</td>
<td>ASSEMBLER'S RAM SPACE</td>
<td>0100 - 0139</td>
</tr>
<tr>
<td>#6</td>
<td>USER LOAD AREA</td>
<td>0000 - 00FF</td>
</tr>
</tbody>
</table>

17.03 I/O VECTOR ASSIGNMENTS:

<table>
<thead>
<tr>
<th>Vector</th>
<th>Editor</th>
<th>Assembler</th>
<th>Cassette</th>
</tr>
</thead>
<tbody>
<tr>
<td>:CI</td>
<td>:SK</td>
<td>:SK</td>
<td>:SK</td>
</tr>
<tr>
<td>:CO</td>
<td>:SV</td>
<td>:SV</td>
<td>:SV</td>
</tr>
<tr>
<td>:OI</td>
<td>:AO</td>
<td>:AO</td>
<td>:I1</td>
</tr>
<tr>
<td>:OO</td>
<td>:AI</td>
<td>:AI</td>
<td>:01</td>
</tr>
<tr>
<td>:SI</td>
<td>:BI</td>
<td>:BO</td>
<td>:I1</td>
</tr>
<tr>
<td>:SO</td>
<td>:SV</td>
<td>:SV</td>
<td>:01</td>
</tr>
</tbody>
</table>
17.04 SAMPLE COMMANDS

.M :SI
.SI :BI :BO ;CHANGE FROM EDITOR TO ASSEMBLER
;CHANGE BACK TO :BI FOR EDITOR

.E E003 ;EXIT TO MONITOR
.E :AS ;EXIT TO ASSEMBLER

.E :ED ;ENTER EDITOR TO START NEW FILE
.E :ER ;REENTER EDITOR TO MODIFY A FILE

.L 0,200 ;CALL LOADER WITH 0 OFFSET TO ORG
;BUILD SYMBOL TABLE AT 200H

LABEL-$ ;RELATIVE ADDRESSING USES -$ ;LABELS DO NOT HAVE A ' : '
CHAPTER 18 --- PLOTTING

18.01 BEAUTIFUL BIRTHDAY PLOTS

THE PROGRAM LISTING USES AN HP7225A PLOTTER WHICH IS VERY INTELLIGENT. HOWEVER, YOU CAN SUBSTITUTE EQUIVALENT PLOTTER CONTROL COMMANDS FOR YOUR PLOTTER.

100 REM --- BIRTHDAY PLOTS ---
110 :
120 CLEAR 150:DIM R(18,18)
130 :
140 REM POKE AN RS232 DRIVER IN MEMORY AND USE IT TO SEND
150 REM PLOTTER CONTROL SEQUENCES VIA PRINT STRINGS.
160 :
170 FOR J=0 TO 14:READ I:POKE J,I:NEXT J
180 DATA 245,245,62,128,211,254,241,205,18
190 DATA 224,205,27,224,241,201
200 :
210 INPUT "NAME";N$
220 INPUT "RANDOM NUMBER";A:J=RND(-A*2-1)
230 :
240 REM FIND MONITOR WORK AREA AND CHANGE OUTPUT VECTOR
250 REM TO POINT TO THE RS232 DRIVER ROUTINE.
260 :
270 M=256*PEEK(-4095)+PEEK(-4096)-65536
280 POKE M-47,0:POKE M-46,0
290 :
300 REM TURN PLOTTER ON, INITIALIZE IT, AND IDENTIFY
310 REM THE PHYSICAL AREA OF THE PLOTTING WINDOW.
320 :
330 PRINT CHR$(27)+".(IN;"
340 PRINT "IP1328,1000,9328,6769;"
350 PRINT "SI.3,.4;" : REM SIZE LETTERING WIDTH AND HEIGHT
360 :
370 REM THIS LOOP SELECTS RANDOM POINTS TO CREATE THE PLOT
380 :
390 B=RND(1)*.1 + .02
400 C=INT(RND(1)*10)+7
410 E=0:D=0:G=1000:F=1000
420 FOR H=1TOC
430 R(H,1)=RND(1)*1000
440 IF R(H,1)<>D THEN 460
450 D=R(H,1)
460 IF R(H,1)>F THEN 480
470 F=R(H,1)
480 R(H,2)=RND(1)*1000
490 IF R(H,2)<E THEN 510
500 E=R(H,2)
510 IF R(H,2)<>G THEN 530
520 G=R(H,2)
530 NEXT H
540 :
550 REM NOW MAP THE LOGICAL WINDOW TO THE PLOTTING SURFACE
560 :
570 PRINT "SC";F:D;G-.1*(E-G);E;";"
580 :
590 REM PLOT THE BIRTHDAY PLOT
600 :
610 PRINT "PA";R(1,1);R(1,2);";PD;"
620 FOR H=1TO50
630 R(C+1,1)=R(1,1)
640 R(C+1,2)=R(1,2)
650 FOR I=1TOC+1
660 PRINT "PA";INT(R(I,1));INT(R(I,2));";"
670 IF I>C THEN 700
680 R(I,1)=B*(R(I+1,1)-R(I,1))+R(I,1)
690 R(I,2)=B*(R(I+1,2)-R(I,2))+R(I,2)
700 NEXT I
710 NEXT H
720 :
730 REM LABEL THE PLOT WITH THE NAME TEXT
740 :
750 PRINT "PU";PA";(D-F)*.5;G-.1*(E-G)";"
760 A$=MID$(STR$(A),2)+"II+N$
770 R=LEN(A$)/2
780 PRINT "CP";R;-1;";"
790 PRINT "LB#";A$;CHR$(3)
800 :
810 REM TURN PLOTTER OFF, AND RESTORE VECTOR TO >SE 0=V
820 :
830 PRINT "IN";CHR$(27)+".)"
840 POKE M-47,27:POKE M-46,224
850 END

SUMMARY OF 7225A PLOTTER CONTROLS USED IN THE EXAMPLE:

**IPx1,y1,x2,y2** - WINDOW CORNER POINTS ON PLOTTER SURFACE
**SCx1,y1,x2,y2** - SCALE LOGICAL CORNER POINTS TO PHYSICAL WINDOW
**PAx,y** - MOVE PEN TO ABSOLUTE LOGICAL POINT OF (x,y)
**PU** - PEN UP
**PD** - PEN DOWN
**CPx,y** - MOVE RELATIVE x CHARACTER WIDTHS, y HEIGHTS
**LB** - LETTER TEXT STRING WHICH FOLLOWS
**SIx,y** - LETTER WIDTH x, HEIGHT y IN CENTIMETERS
18.02 3-D FUNCTION PLOTS USING SHADED PRINT DENSITY

100 REM --- PATTERNS ---
110 :
120 REM PUT FUNCTION TO BE GRAPHED IN FNZ()= ON LINE 140
130 :
140 DEF FNZ(X)=COS(X)*COS(Y) :REM SAMPLE 3-D FUNCTION
150 :
160 L=18:GOSUB 5000
170 INPUT "DOMAIN OF X-AXIS ";X1,X2 :REM TRY -7,7
180 U=(X2-X1)/64
190 INPUT "DOMAIN OF Y-AXIS ";Y1,Y2 :REM TRY -7,7
200 W=-(Y1-Y2)/30
210 INPUT "RANGE OF FUNCTION";R1,R2 :REM TRY -1.01,1.01
220 L=L/(R2-R1):PRINT CHR$(12):Y=Y1
230 FOR J=1 TO 30:X=X1:FOR I=1 TO 64
240 P=INT(L*(FNZ(X)-R1)+192 : POKE -4033+I+J*64,P
250 X=X+U : NEXT I : Y=Y+W : NEXT J
260 GOTO 260 : REM ENDLESS LOOP SO DISPLAY IS NOT RUINED
270 END
5000 REM --- DEFINE GRAPHIC SYMBOLS ---
5010 FOR I=1 TO L*8: READ A: POKE -513+I,A: NEXT I: RETURN
6000 DATA 0,0,0,0,0,0,0,0
6010 DATA 0,0,32,0,0,0,2,0
6020 DATA 0,32,0,4,64,0,8,0
6030 DATA 4,64,16,1,8,128,2,32
6040 DATA 65,16,20,33,132,16,66,8
6050 DATA 146,32,9,128,17,138,16,69
6060 DATA 145,74,17,68,137,18,132,82
6070 DATA 41,138,98,41,146,41,196,41
6080 DATA 85,74,101,146,73,178,74,149
6090 DATA 85,42,83,186,86,170,165,102
6100 DATA 85,229,85,174,85,202,85,171
6110 DATA 93,234,87,186,213,174,117,171
6120 DATA 109,222,245,107,218,183,247,251
6130 DATA 237,119,221,119,237,190,111,219
6140 DATA 190,238,251,222,123,238,190,247
6150 DATA 251,191,238,254,247,127,253,223
6160 DATA 255,223,255,254,255,191,255,251
6170 DATA 255,255,255,255,255,255,255,255
CHAPTER 19 -- TABLES AND FORMS

19.01 BASIC'S TOKENS

<table>
<thead>
<tr>
<th>HEX</th>
<th>-9-</th>
<th>-A-</th>
<th>-B-</th>
<th>-C-</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>END</td>
<td>STOP</td>
<td>PN</td>
<td>INT</td>
</tr>
<tr>
<td>1</td>
<td>FOR</td>
<td>OUT</td>
<td>SPC</td>
<td>ABS</td>
</tr>
<tr>
<td>2</td>
<td>NEXT</td>
<td>ON</td>
<td>THEN</td>
<td>USR</td>
</tr>
<tr>
<td>3</td>
<td>DATA</td>
<td>NULL</td>
<td>NOT</td>
<td>FRE</td>
</tr>
<tr>
<td>4</td>
<td>BYE</td>
<td>WAIT</td>
<td>STEP</td>
<td>INP</td>
</tr>
<tr>
<td>5</td>
<td>INPUT</td>
<td>DEF</td>
<td>+</td>
<td>POS</td>
</tr>
<tr>
<td>6</td>
<td>DIM</td>
<td>POKE</td>
<td>-</td>
<td>SQR</td>
</tr>
<tr>
<td>7</td>
<td>READ</td>
<td>PRINT</td>
<td>*</td>
<td>RND</td>
</tr>
<tr>
<td>8</td>
<td>LET</td>
<td>CONT</td>
<td>/</td>
<td>LOG</td>
</tr>
<tr>
<td>9</td>
<td>GOTO</td>
<td>LIST</td>
<td>&lt;</td>
<td>EXP</td>
</tr>
<tr>
<td>A</td>
<td>RUN</td>
<td>CLEAR</td>
<td>AND</td>
<td>COS</td>
</tr>
<tr>
<td>B</td>
<td>IF</td>
<td>CLOAD</td>
<td>OR</td>
<td>SIN</td>
</tr>
<tr>
<td>C</td>
<td>RESTORE</td>
<td>CSAVE</td>
<td>&gt;</td>
<td>TAN</td>
</tr>
<tr>
<td>D</td>
<td>GOSUB</td>
<td>NEW</td>
<td>=</td>
<td>ATN</td>
</tr>
<tr>
<td>E</td>
<td>RETURN</td>
<td>TAB</td>
<td>&lt;</td>
<td>PEEK</td>
</tr>
<tr>
<td>F</td>
<td>REM</td>
<td>TO</td>
<td>SGN</td>
<td>LEN</td>
</tr>
</tbody>
</table>

EXAMPLE: 8F - REM  BE - PEEK  C6 - MID$

===============================================

19.02 HEXADECIMAL - BINARY CONVERSION TABLE

| 0   | 0000 | 8  | 1000 |
| 1   | 0001 | 9  | 1001 |
| 2   | 0010 | A  | 1010 |
| 3   | 0011 | B  | 1011 |
| 4   | 0100 | C  | 1100 |
| 5   | 0101 | D  | 1101 |
| 6   | 0110 | E  | 1110 |
| 7   | 0111 | F  | 1111 |

===============================================

19.03 POWERS OF 2 TABLE: 2 ^ N

| 0   | 1   | 9  | 512  |
| 1   | 2   | 10 | 1024 |
| 2   | 4   | 11 | 2048 |
| 3   | 8   | 12 | 4096 |
| 4   | 16  | 13 | 8192 |
| 5   | 32  | 14 | 16384|
| 6   | 64  | 15 | 32768|
| 7   | 128 | 16 | 65536|
| 8   | 256 | 17 | 131072|
### 19.04 DECIMAL - HEXADECIMAL CONVERSION TABLE

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
</tr>
<tr>
<td>32</td>
<td>20</td>
</tr>
<tr>
<td>33</td>
<td>21</td>
</tr>
<tr>
<td>34</td>
<td>22</td>
</tr>
<tr>
<td>35</td>
<td>23</td>
</tr>
<tr>
<td>36</td>
<td>24</td>
</tr>
<tr>
<td>37</td>
<td>25</td>
</tr>
<tr>
<td>38</td>
<td>26</td>
</tr>
<tr>
<td>39</td>
<td>27</td>
</tr>
<tr>
<td>40</td>
<td>28</td>
</tr>
<tr>
<td>41</td>
<td>29</td>
</tr>
<tr>
<td>42</td>
<td>2A</td>
</tr>
<tr>
<td>43</td>
<td>2B</td>
</tr>
<tr>
<td>44</td>
<td>2C</td>
</tr>
<tr>
<td>45</td>
<td>2D</td>
</tr>
<tr>
<td>46</td>
<td>2E</td>
</tr>
<tr>
<td>47</td>
<td>2F</td>
</tr>
</tbody>
</table>
### 19.05 Control Codes and Their Functions

<table>
<thead>
<tr>
<th>HEX</th>
<th>DEC</th>
<th>Control</th>
<th>Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>@</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>A</td>
<td>SOH</td>
<td>CURSOR LEFT</td>
</tr>
<tr>
<td>02</td>
<td>2</td>
<td>B</td>
<td>STX</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>3</td>
<td>C</td>
<td>ETX</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>4</td>
<td>D</td>
<td>EOT</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>5</td>
<td>E</td>
<td>ENQ</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>6</td>
<td>F</td>
<td>ACK</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>7</td>
<td>G</td>
<td>BELL</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>8</td>
<td>H</td>
<td>BS</td>
<td>RUBOUT</td>
</tr>
<tr>
<td>09</td>
<td>9</td>
<td>I</td>
<td>HT</td>
<td></td>
</tr>
<tr>
<td>0A</td>
<td>10</td>
<td>J</td>
<td>LF</td>
<td>LINE FEED</td>
</tr>
<tr>
<td>0B</td>
<td>11</td>
<td>K</td>
<td>VT</td>
<td></td>
</tr>
<tr>
<td>0C</td>
<td>12</td>
<td>L</td>
<td>FF</td>
<td>CLEAR SCREEN</td>
</tr>
<tr>
<td>0D</td>
<td>13</td>
<td>&lt;CR&gt;</td>
<td>CR</td>
<td>RETURN</td>
</tr>
<tr>
<td>0E</td>
<td>14</td>
<td>N</td>
<td>SO</td>
<td></td>
</tr>
<tr>
<td>0F</td>
<td>15</td>
<td>O</td>
<td>SI</td>
<td>STOP INPUT</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>P</td>
<td>DLE</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>17</td>
<td>Q</td>
<td>DC1</td>
<td>HOME</td>
</tr>
<tr>
<td>12</td>
<td>18</td>
<td>R</td>
<td>DC2</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>19</td>
<td>S</td>
<td>DC3</td>
<td>CURSOR RIGHT</td>
</tr>
<tr>
<td>14</td>
<td>20</td>
<td>T</td>
<td>DC4</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>21</td>
<td>U</td>
<td>NAK</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>22</td>
<td>V</td>
<td>SYN</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>23</td>
<td>W</td>
<td>ETB</td>
<td>CURSOR UP</td>
</tr>
<tr>
<td>18</td>
<td>24</td>
<td>X</td>
<td>CAN</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>25</td>
<td>Y</td>
<td>EM</td>
<td></td>
</tr>
<tr>
<td>1A</td>
<td>26</td>
<td>Z</td>
<td>SUB</td>
<td>CURSOR DOWN</td>
</tr>
<tr>
<td>1B</td>
<td>27</td>
<td>[</td>
<td>ESC</td>
<td>ESCAPE</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>\</td>
<td>FS</td>
<td></td>
</tr>
<tr>
<td>1D</td>
<td>29</td>
<td>]</td>
<td>GS</td>
<td></td>
</tr>
<tr>
<td>1E</td>
<td>30</td>
<td>^</td>
<td>RS</td>
<td></td>
</tr>
<tr>
<td>1F</td>
<td>31</td>
<td>-</td>
<td>US</td>
<td></td>
</tr>
</tbody>
</table>

Functions accessed using `CHR$( )`.  

Example: 100 PRINT CHR$(12);:REM clears screen
19.07 MACHINE LANGUAGE CODING FORM

ROUTINE NAME:                          PAGE  OF
DESCRIPTION:                          DATE

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>OBJECT CODE</th>
<th>LABEL</th>
<th>INSTRUCTION</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
<td>---------</td>
</tr>
</tbody>
</table>
### 19.08 SERIAL INTERFACE PINOUTS

<table>
<thead>
<tr>
<th>PIN#</th>
<th>SIGNAL</th>
<th>PIN#</th>
<th>SIGNAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SHIELD 1</td>
<td>14</td>
<td>SHIELD 2</td>
</tr>
<tr>
<td>2</td>
<td>RS232 OUT</td>
<td>15</td>
<td>MIC 1</td>
</tr>
<tr>
<td>3</td>
<td>RS232 IN</td>
<td>16</td>
<td>MIC 2</td>
</tr>
<tr>
<td>4</td>
<td>GROUND</td>
<td>17</td>
<td>GROUND</td>
</tr>
<tr>
<td>5</td>
<td>AUX 1</td>
<td>18</td>
<td>AUX 2</td>
</tr>
<tr>
<td>6</td>
<td>GROUND</td>
<td>19</td>
<td>GROUND</td>
</tr>
<tr>
<td>7</td>
<td>GROUND</td>
<td>20</td>
<td>EAR 1</td>
</tr>
<tr>
<td>8</td>
<td>GROUND</td>
<td>21</td>
<td>EAR 2</td>
</tr>
<tr>
<td>9</td>
<td>+12 VOLTS</td>
<td>22</td>
<td>UNUSED</td>
</tr>
<tr>
<td>10</td>
<td>UNUSED</td>
<td>23</td>
<td>RS232 OUT</td>
</tr>
<tr>
<td>11</td>
<td>RS232 IN</td>
<td>24</td>
<td>MOTOR #1-</td>
</tr>
<tr>
<td>12</td>
<td>MOTOR #1+</td>
<td>25</td>
<td>MOTOR #2-</td>
</tr>
<tr>
<td>13</td>
<td>MOTOR #2+</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 19.09 PARALLEL INTERFACE PINOUTS

<table>
<thead>
<tr>
<th>PIN#</th>
<th>SIGNAL</th>
<th>PIN#</th>
<th>SIGNAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GROUND</td>
<td>14</td>
<td>UNUSED</td>
</tr>
<tr>
<td>2</td>
<td>OUTPUT DATA ACCEPTED</td>
<td>15</td>
<td>+5 VOLTS</td>
</tr>
<tr>
<td>3</td>
<td>OUTPUT DATA AVAILABLE</td>
<td>16</td>
<td>OUTPUT BIT 0</td>
</tr>
<tr>
<td>4</td>
<td>OUTPUT BIT 7</td>
<td>17</td>
<td>OUTPUT BIT 1</td>
</tr>
<tr>
<td>5</td>
<td>OUTPUT BIT 6</td>
<td>18</td>
<td>OUTPUT BIT 2</td>
</tr>
<tr>
<td>6</td>
<td>OUTPUT BIT 5</td>
<td>19</td>
<td>OUTPUT BIT 3</td>
</tr>
<tr>
<td>7</td>
<td>OUTPUT BIT 4</td>
<td>20</td>
<td>+5 VOLTS</td>
</tr>
<tr>
<td>8</td>
<td>GROUND</td>
<td>21</td>
<td>INPUT DATA ACCEPTED</td>
</tr>
<tr>
<td>9</td>
<td>INPUT DATA AVAILABLE</td>
<td>22</td>
<td>INPUT BIT 1</td>
</tr>
<tr>
<td>10</td>
<td>INPUT BIT 0</td>
<td>23</td>
<td>INPUT BIT 3</td>
</tr>
<tr>
<td>11</td>
<td>INPUT BIT 2</td>
<td>24</td>
<td>INPUT BIT 5</td>
</tr>
<tr>
<td>12</td>
<td>INPUT BIT 4</td>
<td>25</td>
<td>INPUT BIT 7</td>
</tr>
<tr>
<td>13</td>
<td>INPUT BIT 6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 19.10 DATA PORT ASSIGNMENTS

<table>
<thead>
<tr>
<th>PORT</th>
<th>BIT</th>
<th>INPUT FUNCTION</th>
<th>OUTPUT FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPCH</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>252</td>
<td>2</td>
<td>SERIAL</td>
<td>SERIAL</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>INPUT</td>
<td>OUTPUT</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>DATA</td>
<td>DATA</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>BYTE</td>
<td>BYTE</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0FDH</td>
<td>0</td>
<td>SERIAL OUT BUFFER EMPTY</td>
<td>BITS PER CHARACTER NB1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>SERIAL DATA AVAILABLE</td>
<td>BITS PER CHARACTER NB2</td>
</tr>
<tr>
<td>253</td>
<td>2</td>
<td>OVER-RUN</td>
<td>NUMBER OF STOP BITS</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>FRAMING ERROR</td>
<td>PARITY SELECT</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>PARITY ERROR</td>
<td>NO PARITY</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>UNUSED</td>
<td>UNUSED</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>UNUSED</td>
<td>UNUSED</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>UNUSED</td>
<td>UNUSED</td>
</tr>
<tr>
<td>0FEH</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>KEYBOARD</td>
<td>KEYBOARD</td>
</tr>
<tr>
<td>254</td>
<td>2</td>
<td>DATA</td>
<td>SCAN COLUMN</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>BYTE (5 BITS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>HORIZONTAL SYNC PULSE</td>
<td>MOTOR #1 CONTROL</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>PARALLEL OUTPUT READ</td>
<td>MOTOR #2 CONTROL</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>PARALLEL INPUT AVAILABLE</td>
<td>BAUD RATE SELECT</td>
</tr>
<tr>
<td>0FFH</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>255</td>
<td>2</td>
<td>PARALLEL</td>
<td>PARALLEL</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>INPUT</td>
<td>OUTPUT</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>DATA</td>
<td>DATA</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>BYTE</td>
<td>BYTE</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## 19.11 ASCII CHARACTER SET

<table>
<thead>
<tr>
<th>MSB</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSB</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>0</td>
<td>NUL</td>
<td>DLE</td>
<td>SPACE</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>`</td>
<td>p</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>&quot;</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td>ENG</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>AMP</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
</tr>
<tr>
<td>7</td>
<td>BELL</td>
<td>ETB</td>
<td>'</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
</tr>
<tr>
<td>A</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>VT</td>
<td>ESC</td>
<td>;</td>
<td>K</td>
<td>[</td>
<td>k</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>M</td>
<td>]</td>
<td>m</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>SO</td>
<td>RS</td>
<td>.</td>
<td>N</td>
<td>~</td>
<td>n</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>SI</td>
<td>VS</td>
<td>/</td>
<td>O</td>
<td>UNDER</td>
<td>o</td>
<td>RUB</td>
<td></td>
</tr>
</tbody>
</table>

### MSB, 8

| LSB  | ------| ------| ------| ------| ------| ------| ------| ------|
| 0    | GR   | 1     | GR    | J     | GR    | /     | GS    | 1     |
| 1    | GR   | 2     | GR    | K     | GR    | K-    | GS    | 2     |
| 2    | GR   | 3     | GR    | T     | GR    | L     | GR    | K7    |
| 3    | GR   | 4     | GR    | Y     | GR    | ;     | GR    | K8    |
| 4    | GR   | 5     | GR    | U     | GR    | @     | GR    | K9    |
| 5    | GR   | 6     | GR    | I     | GR    | \    | GS    | 6     |
| 6    | GR   | 7     | GR    | O     | GR    | RUB   | GR    | K4    |
| 7    | GR   | 8     | GR    | P     | GR    | Z     | GR    | K6    |
| 8    | GR   | 9     | GR    | X     | GR    | K*    | GS    | 9     |
| 9    | GR   | 0     | GR    | C     | GR    | K1    | GS    | 0     |
| A    | GR   | 1     | GR    | A     | GR    | V     | GR    | K2    |
| B    | GR   | 2     | GR    | S     | GR    | B     | GR    | K3    |
| C    | GR   | 3     | GR    | D     | GR    | N     | GR    | K+    |
| D    | GR   | 4     | GR    | T     | GR    | M     | GR    | K0    |
| E    | GR   | 5     | GR    | Q     | GR    | R     | GR    | K    |
| F    | GR   | 6     | GR    | W     | GR    | H     | GR    | K    |

**KEY:** GR = GRAPHIC GS = GRAPHIC AND SHIFT K1 = KEYPAD 1 MSB= MOST SIGNIFICANT BYTE LSB= LEAST SIGNIFICANT BYTE

**EXAMPLE:** GRAPHIC SHIFT TAB = CD HEX (205 DECIMAL)
19.12 KEYBOARD STRUCTURE: OUT (OFEH), nn AND IN A, (OFEH)

The key is depressed when the bit is a logical low.

<table>
<thead>
<tr>
<th>OUT nn</th>
<th>BIT 0</th>
<th>BIT 1</th>
<th>BIT 2</th>
<th>BIT 3</th>
<th>BIT 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>STOP</td>
<td>GRAPHIC</td>
<td>CTRL</td>
<td>SHFTLOCK</td>
<td>SHIFT</td>
</tr>
<tr>
<td>1</td>
<td>CLEAR</td>
<td>REPEAT</td>
<td>SPACE</td>
<td>SKIP</td>
<td>ESC</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>Z</td>
<td>A</td>
<td>Q</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>D</td>
<td>S</td>
<td>W</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>R</td>
<td>E</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>V</td>
<td>G</td>
<td>T</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>M</td>
<td>N</td>
<td>H</td>
<td>Y</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>K</td>
<td>I</td>
<td>J</td>
<td>U</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>,</td>
<td>L</td>
<td>O</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>/</td>
<td>.</td>
<td>;</td>
<td>P</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>\</td>
<td>@</td>
<td>[</td>
<td>:</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>UNDERLINE</td>
<td>RETURN</td>
<td>LINEFEED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>KEYPAD +</td>
<td>KEYPAD *</td>
<td>KEYPAD /</td>
<td>KEYPAD -</td>
<td>NOT USED</td>
</tr>
<tr>
<td>13</td>
<td>KEYPAD 0</td>
<td>KEYPAD 1</td>
<td>KEYPAD 4</td>
<td>KEYPAD 8</td>
<td>KEYPAD 7</td>
</tr>
<tr>
<td>14</td>
<td>KEYPAD .</td>
<td>KEYPAD 2</td>
<td>KEYPAD 5</td>
<td>KEYPAD 6</td>
<td>KEYPAD 9</td>
</tr>
<tr>
<td>15</td>
<td>NOT USED</td>
<td>NOT USED</td>
<td>NOT USED</td>
<td>KEYPAD =</td>
<td>KEYPAD 3</td>
</tr>
</tbody>
</table>

19.13 GRAPHIC CHARACTER STARTING ADDRESSES

<table>
<thead>
<tr>
<th>ASC</th>
<th>KEY</th>
<th>HEX</th>
<th>DEC</th>
<th>ASC</th>
<th>KEY</th>
<th>HEX</th>
<th>DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>GR 1</td>
<td>FC00</td>
<td>-1024</td>
<td>192</td>
<td>GS 1</td>
<td>FE00</td>
<td>-512</td>
</tr>
<tr>
<td>129</td>
<td>GR 2</td>
<td>FC08</td>
<td>-1016</td>
<td>193</td>
<td>GS 2</td>
<td>FE08</td>
<td>-504</td>
</tr>
<tr>
<td>130</td>
<td>GR 3</td>
<td>FC10</td>
<td>-1008</td>
<td>194</td>
<td>GS 3</td>
<td>FE10</td>
<td>-496</td>
</tr>
<tr>
<td>131</td>
<td>GR 4</td>
<td>FC18</td>
<td>-1000</td>
<td>195</td>
<td>GS 4</td>
<td>FE18</td>
<td>-488</td>
</tr>
<tr>
<td>132</td>
<td>GR 5</td>
<td>FC20</td>
<td>-992</td>
<td>196</td>
<td>GS 5</td>
<td>FE20</td>
<td>-480</td>
</tr>
<tr>
<td>133</td>
<td>GR 6</td>
<td>FC28</td>
<td>-984</td>
<td>197</td>
<td>GS 6</td>
<td>FE28</td>
<td>-472</td>
</tr>
<tr>
<td>134</td>
<td>GR 7</td>
<td>FC30</td>
<td>-976</td>
<td>198</td>
<td>GS 7</td>
<td>FE30</td>
<td>-464</td>
</tr>
<tr>
<td>135</td>
<td>GR 8</td>
<td>FC38</td>
<td>-968</td>
<td>199</td>
<td>GS 8</td>
<td>FE38</td>
<td>-456</td>
</tr>
<tr>
<td>136</td>
<td>GR 9</td>
<td>FC40</td>
<td>-960</td>
<td>200</td>
<td>GS 9</td>
<td>FE40</td>
<td>-448</td>
</tr>
<tr>
<td>137</td>
<td>GR 0</td>
<td>FC48</td>
<td>-952</td>
<td>201</td>
<td>GS 0</td>
<td>FE48</td>
<td>-440</td>
</tr>
<tr>
<td>138</td>
<td>GR :</td>
<td>FC50</td>
<td>-944</td>
<td>202</td>
<td>GS :</td>
<td>FE50</td>
<td>-432</td>
</tr>
<tr>
<td>139</td>
<td>GR -</td>
<td>FC58</td>
<td>-936</td>
<td>203</td>
<td>GS -</td>
<td>FE58</td>
<td>-424</td>
</tr>
<tr>
<td>140</td>
<td>GR ^</td>
<td>FC60</td>
<td>-928</td>
<td>204</td>
<td>GS ^</td>
<td>FE60</td>
<td>-416</td>
</tr>
<tr>
<td>141</td>
<td>GR TAB</td>
<td>FC68</td>
<td>-920</td>
<td>205</td>
<td>GS TAB</td>
<td>FE68</td>
<td>-408</td>
</tr>
<tr>
<td>142</td>
<td>GR Q</td>
<td>FC70</td>
<td>-912</td>
<td>206</td>
<td>GS Q</td>
<td>FE70</td>
<td>-400</td>
</tr>
<tr>
<td>143</td>
<td>GR W</td>
<td>FC78</td>
<td>-904</td>
<td>207</td>
<td>GS W</td>
<td>FE78</td>
<td>-392</td>
</tr>
<tr>
<td>144</td>
<td>GR E</td>
<td>FC80</td>
<td>-896</td>
<td>208</td>
<td>GS E</td>
<td>FE80</td>
<td>-384</td>
</tr>
<tr>
<td>145</td>
<td>GR R</td>
<td>FC88</td>
<td>-888</td>
<td>209</td>
<td>GS R</td>
<td>FE88</td>
<td>-376</td>
</tr>
<tr>
<td>146</td>
<td>GR T</td>
<td>FC90</td>
<td>-880</td>
<td>210</td>
<td>GS T</td>
<td>FE90</td>
<td>-368</td>
</tr>
<tr>
<td>147</td>
<td>GR Y</td>
<td>FC98</td>
<td>-872</td>
<td>211</td>
<td>GS Y</td>
<td>FE98</td>
<td>-360</td>
</tr>
<tr>
<td>148</td>
<td>GR U</td>
<td>FCA0</td>
<td>-864</td>
<td>212</td>
<td>GS U</td>
<td>FEA0</td>
<td>-352</td>
</tr>
<tr>
<td>149</td>
<td>GR I</td>
<td>FCA8</td>
<td>-856</td>
<td>213</td>
<td>GS I</td>
<td>FEA8</td>
<td>-344</td>
</tr>
<tr>
<td>150</td>
<td>GR O</td>
<td>FCB0</td>
<td>-848</td>
<td>214</td>
<td>GS O</td>
<td>FEB0</td>
<td>-336</td>
</tr>
<tr>
<td>151</td>
<td>GR P</td>
<td>FCB8</td>
<td>-840</td>
<td>215</td>
<td>GS P</td>
<td>FE88</td>
<td>-328</td>
</tr>
<tr>
<td>152</td>
<td>GR [</td>
<td>FCC0</td>
<td>-832</td>
<td>216</td>
<td>GS [</td>
<td>FEC0</td>
<td>-320</td>
</tr>
<tr>
<td>153</td>
<td>GR H</td>
<td>FC88</td>
<td>-824</td>
<td>217</td>
<td>GS H</td>
<td>FEC8</td>
<td>-312</td>
</tr>
<tr>
<td>154</td>
<td>GR A</td>
<td>FCD0</td>
<td>-816</td>
<td>218</td>
<td>GS A</td>
<td>PED0</td>
<td>-304</td>
</tr>
<tr>
<td>155</td>
<td>GR S</td>
<td>FCD8</td>
<td>-808</td>
<td>219</td>
<td>GS S</td>
<td>PED8</td>
<td>-296</td>
</tr>
<tr>
<td>156</td>
<td>GR D</td>
<td>FCE0</td>
<td>-800</td>
<td>220</td>
<td>GS D</td>
<td>FEE0</td>
<td>-288</td>
</tr>
<tr>
<td>157</td>
<td>GR F</td>
<td>FCE8</td>
<td>-792</td>
<td>221</td>
<td>GS F</td>
<td>FEE8</td>
<td>-280</td>
</tr>
<tr>
<td>158</td>
<td>GR G</td>
<td>FC80</td>
<td>-784</td>
<td>222</td>
<td>GS G</td>
<td>FEF0</td>
<td>-272</td>
</tr>
<tr>
<td>159</td>
<td>GR J</td>
<td>FD00</td>
<td>-768</td>
<td>224</td>
<td>GS J</td>
<td>FF00</td>
<td>-252</td>
</tr>
<tr>
<td>160</td>
<td>GR K</td>
<td>FD08</td>
<td>-760</td>
<td>225</td>
<td>GS K</td>
<td>FF08</td>
<td>-244</td>
</tr>
<tr>
<td>161</td>
<td>GR L</td>
<td>FD10</td>
<td>-752</td>
<td>226</td>
<td>GS L</td>
<td>FF10</td>
<td>-236</td>
</tr>
<tr>
<td>162</td>
<td>GR ;</td>
<td>FD18</td>
<td>-744</td>
<td>227</td>
<td>GS ;</td>
<td>FF18</td>
<td>-228</td>
</tr>
<tr>
<td>163</td>
<td>GR @</td>
<td>FD20</td>
<td>-736</td>
<td>228</td>
<td>GS @</td>
<td>FF20</td>
<td>-220</td>
</tr>
<tr>
<td>164</td>
<td>GR \</td>
<td>FD28</td>
<td>-728</td>
<td>229</td>
<td>GS \</td>
<td>FF28</td>
<td>-212</td>
</tr>
<tr>
<td>165</td>
<td>GR RUB</td>
<td>FD30</td>
<td>-720</td>
<td>230</td>
<td>GS RUB</td>
<td>FF30</td>
<td>-204</td>
</tr>
<tr>
<td>166</td>
<td>GR Z</td>
<td>FD38</td>
<td>-712</td>
<td>231</td>
<td>GS Z</td>
<td>FF38</td>
<td>-196</td>
</tr>
<tr>
<td>167</td>
<td>GR X</td>
<td>FD40</td>
<td>-704</td>
<td>232</td>
<td>GS X</td>
<td>FF40</td>
<td>-188</td>
</tr>
<tr>
<td>168</td>
<td>GR C</td>
<td>FD48</td>
<td>-696</td>
<td>233</td>
<td>GS C</td>
<td>FF48</td>
<td>-180</td>
</tr>
<tr>
<td>169</td>
<td>GR V</td>
<td>FD50</td>
<td>-688</td>
<td>234</td>
<td>GS V</td>
<td>FF50</td>
<td>-172</td>
</tr>
<tr>
<td>170</td>
<td>GR B</td>
<td>FD58</td>
<td>-680</td>
<td>235</td>
<td>GS B</td>
<td>FF58</td>
<td>-168</td>
</tr>
<tr>
<td>171</td>
<td>GR N</td>
<td>FD60</td>
<td>-672</td>
<td>236</td>
<td>GS N</td>
<td>FF60</td>
<td>-160</td>
</tr>
<tr>
<td>172</td>
<td>GR M</td>
<td>FD68</td>
<td>-664</td>
<td>237</td>
<td>GS M</td>
<td>FF68</td>
<td>-152</td>
</tr>
<tr>
<td>173</td>
<td>GR ,</td>
<td>FD70</td>
<td>-656</td>
<td>238</td>
<td>GS ,</td>
<td>FF70</td>
<td>-144</td>
</tr>
<tr>
<td>174</td>
<td>GR .</td>
<td>FD78</td>
<td>-648</td>
<td>239</td>
<td>GS .</td>
<td>FF78</td>
<td>-136</td>
</tr>
<tr>
<td>175</td>
<td>GR /</td>
<td>FD80</td>
<td>-640</td>
<td>240</td>
<td>GS /</td>
<td>FF80</td>
<td>-128</td>
</tr>
<tr>
<td>176</td>
<td>GR K-</td>
<td>FD88</td>
<td>-632</td>
<td>241</td>
<td>GS K-</td>
<td>FF88</td>
<td>-120</td>
</tr>
<tr>
<td>177</td>
<td>GR K7</td>
<td>FD90</td>
<td>-624</td>
<td>242</td>
<td>GS K7</td>
<td>FF90</td>
<td>-112</td>
</tr>
<tr>
<td>178</td>
<td>GR K8</td>
<td>FD98</td>
<td>-616</td>
<td>243</td>
<td>GS K8</td>
<td>FF98</td>
<td>-104</td>
</tr>
<tr>
<td>179</td>
<td>GR K9</td>
<td>FDA0</td>
<td>-608</td>
<td>244</td>
<td>GS K9</td>
<td>FFA0</td>
<td>-96</td>
</tr>
<tr>
<td>180</td>
<td>GR K/</td>
<td>FDA8</td>
<td>-600</td>
<td>245</td>
<td>GS K/</td>
<td>FFA8</td>
<td>-88</td>
</tr>
<tr>
<td>181</td>
<td>GR K4</td>
<td>FDB0</td>
<td>-592</td>
<td>246</td>
<td>GS K4</td>
<td>FFB0</td>
<td>-80</td>
</tr>
<tr>
<td>182</td>
<td>GR K6</td>
<td>FDB8</td>
<td>-584</td>
<td>247</td>
<td>GS K6</td>
<td>FFB8</td>
<td>-72</td>
</tr>
<tr>
<td>183</td>
<td>GR K*</td>
<td>FDC0</td>
<td>-576</td>
<td>248</td>
<td>GS K*</td>
<td>FFC0</td>
<td>-64</td>
</tr>
<tr>
<td>184</td>
<td>GR K1</td>
<td>FDC8</td>
<td>-568</td>
<td>249</td>
<td>GS K1</td>
<td>FFC8</td>
<td>-56</td>
</tr>
<tr>
<td>185</td>
<td>GR K2</td>
<td>FDD0</td>
<td>-560</td>
<td>250</td>
<td>GS K2</td>
<td>FFD0</td>
<td>-48</td>
</tr>
<tr>
<td>186</td>
<td>GR K3</td>
<td>FDD8</td>
<td>-552</td>
<td>251</td>
<td>GS K3</td>
<td>FFD8</td>
<td>-40</td>
</tr>
<tr>
<td>187</td>
<td>GR K+</td>
<td>FDE0</td>
<td>-544</td>
<td>252</td>
<td>GS K+</td>
<td>FFE0</td>
<td>-32</td>
</tr>
<tr>
<td>188</td>
<td>GR K0</td>
<td>FDE8</td>
<td>-536</td>
<td>253</td>
<td>GS K0</td>
<td>FFE8</td>
<td>-24</td>
</tr>
<tr>
<td>190</td>
<td>GR K.</td>
<td>FDF0</td>
<td>-528</td>
<td>254</td>
<td>GS K.</td>
<td>FFF0</td>
<td>-16</td>
</tr>
<tr>
<td>191</td>
<td>GR K=</td>
<td>FDF8</td>
<td>-520</td>
<td>255</td>
<td>GS K=</td>
<td>FFF8</td>
<td>-8</td>
</tr>
</tbody>
</table>

KEY: GR = GRAPHIC  GS = GRAPHIC AND SHIFT  K1 = KEYPAD 1
### 19.14 ONE-STROKE BASIC COMMANDS

<table>
<thead>
<tr>
<th>ASC</th>
<th>KEYS</th>
<th>COMMAND</th>
<th>ASC</th>
<th>KEYS</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>GR</td>
<td>1 END</td>
<td>192</td>
<td>GS</td>
<td>1 STR$</td>
</tr>
<tr>
<td>129</td>
<td>GR</td>
<td>2 FOR</td>
<td>193</td>
<td>GS</td>
<td>2 VAL</td>
</tr>
<tr>
<td>130</td>
<td>GR</td>
<td>3 NEXT</td>
<td>194</td>
<td>GS</td>
<td>3 ASC</td>
</tr>
<tr>
<td>131</td>
<td>GR</td>
<td>4 DATA</td>
<td>195</td>
<td>GS</td>
<td>4 CHR$</td>
</tr>
<tr>
<td>132</td>
<td>GR</td>
<td>5 BYE 0</td>
<td>196</td>
<td>GS</td>
<td>5 LEFT$</td>
</tr>
<tr>
<td>133</td>
<td>GR</td>
<td>6 INPUT</td>
<td>197</td>
<td>GS</td>
<td>6 RIGHT$</td>
</tr>
<tr>
<td>134</td>
<td>GR</td>
<td>7 DIM</td>
<td>198</td>
<td>GS</td>
<td>7 MID$</td>
</tr>
<tr>
<td>135</td>
<td>GR</td>
<td>8 READ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>GR</td>
<td>9 LET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>GR</td>
<td>0 GOTO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>GR</td>
<td>: RUN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>GR</td>
<td>- IF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>GR</td>
<td>^ RESTORE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>GR</td>
<td>TAB GOSUB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>GR</td>
<td>Q RETURN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>GR</td>
<td>W REM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>GR</td>
<td>E STOP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>GR</td>
<td>R OUT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>146</td>
<td>GR</td>
<td>T ON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>147</td>
<td>GR</td>
<td>Y NULL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>148</td>
<td>GR</td>
<td>U WAIT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>GR</td>
<td>I DEF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>GR</td>
<td>O POKE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>GR</td>
<td>P PRINT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>GR</td>
<td>[ CONT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>GR</td>
<td>] LIST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>GR</td>
<td>A CLEAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>GR</td>
<td>S CLOAD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>GR</td>
<td>D CSAVE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>157</td>
<td>GR</td>
<td>F NEW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>GR</td>
<td>G TAB(</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>GR</td>
<td>H TO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>GR</td>
<td>J FN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>GR</td>
<td>K SPC(</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>162</td>
<td>GR</td>
<td>L THEN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>163</td>
<td>GR</td>
<td>; NOT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>164</td>
<td>GR</td>
<td>@ STEP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>165</td>
<td>GR</td>
<td>\ +</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>166</td>
<td>GR</td>
<td>RUB -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>167</td>
<td>GR</td>
<td>Z .</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>168</td>
<td>GR</td>
<td>X /</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>169</td>
<td>GR</td>
<td>C ^</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>GR</td>
<td>V AND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>171</td>
<td>GR</td>
<td>B OR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>172</td>
<td>GR</td>
<td>N &gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>173</td>
<td>GR</td>
<td>M =</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>174</td>
<td>GR</td>
<td>, &lt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>GR</td>
<td>. SGN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>176</td>
<td>GR</td>
<td>/ INT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>177</td>
<td>GR</td>
<td>K - ABS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>178</td>
<td>GR</td>
<td>K7 USR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>GR</td>
<td>K8 FRE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>GR</td>
<td>K9 INP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>181</td>
<td>GR</td>
<td>K/ POS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>GR</td>
<td>K4 SQR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>GR</td>
<td>K6 RND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>GR</td>
<td>K* LOG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>185</td>
<td>GR</td>
<td>K1 EXP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>186</td>
<td>GR</td>
<td>K2 COS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>187</td>
<td>GR</td>
<td>K3 SIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>188</td>
<td>GR</td>
<td>K+ TAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189</td>
<td>GR</td>
<td>K0 ATN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>GR</td>
<td>K. Peek</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>191</td>
<td>GR</td>
<td>K= LEN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**KEY:** GR = GRAPHIC KEY  GS = GRAPHIC AND SHIFT KEYS
ASTRO ATTACKER $21.95

is similar to the arcade game called "ASTRO BLASTER". This action game for the Sorcerer is far superior to all other Sorcerer games because of its high resolution graphics, sound, variety and playability. Astro Attacker graphics are extremely advanced. The display is of the console inside your astro fighter craft. In your console window you see the enemy ships placed against a background of continuously moving stars. Gauges also indicate the amount of fuel remaining and the temperature of your lazer cannons. If you fire too frequently you can overheat the lazers, or if you move recklessly you may run out of fuel. Your challenge is to survive and destroy the Spinners, the Lazer Ships, the Rockets, the Flame Throwers, and the Meteor shower. Docking with the mother ship is crucial to survival as this restores your shield strength and fuel, and cools your lazer cannon. With each succeeding level of play, survival becomes more difficult as the enemy ships attack with greater frequency and quickness. Superb sound too.

Please let us know if you should not be receiving this catalog so we can remove your name from our mailing list.

LIMITED WARRANTY
SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. IN NO EVENT WILL ENSIGN SOFTWARE BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE ANY OF OUR PRODUCTS.

YOU ASSUME RESPONSIBILITY FOR THE SELECTION OF THE PRODUCTS TO ACHIEVE YOUR INTENDED RESULTS, AND FOR THE INSTALLATION, REPAIR, USE AND RESULTS OBTAINED FROM THE PRODUCTS.

HOW TO ORDER:
ALL PRICES ARE IN U.S. DOLLARS

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>OVERSEAS POSTAGE: ADD $3</td>
<td>NO POSTAGE CHARGE IN USA &amp; CANADA</td>
</tr>
</tbody>
</table>

ENCLOSE:
(A) A CHECK IN U.S. DOLLARS DRAWN ON A U.S. BANK, MONEY ORDER OR CASH FOR THE ABOVE AMOUNT, OR
(B) CREDIT CARD, EXPIRATION DATE

NAME: ........................................
STREET: ....................................
CITY/STATE: ................................
POSTCODE: .................. COUNTRY: ............

PLEASE DESCRIBE YOUR COMPUTER CONFIGURATION

MAIL YOUR ORDER TO:
ENSIGN SOFTWARE
2312 N. COLE RD., SUITE E
BOISE, IDAHO 83704 U.S.A.

PHONE: (208) 378-8086

AUTHORS: We seek excellent programs to market worldwide. Our royalties are generous, and our interest in having satisfied customers is keen. We invite you to join with us and our distributors to bring your product into the marketplace. Submit programs for evaluation along with documentation and your phone number.

ENSIGN SOFTWARE
2312 N. Cole Rd., Suite E, Boise, Idaho 83704 U.S.A.